

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS

Совершенствование метода синтеза на основе машинного обучения в САПР
специализированных вычислителей

Обучающийся / Student Бураков Илья Алексеевич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42142

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Системное и прикладное программное обеспечение 2021

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Руководитель ВКР/ Thesis supervisor Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

Документ подписан	
Бураков Илья Алексеевич	
04.06.2023	

(эл. подпись/ signature)

Бураков Илья
Алексеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
04.06.2023	

(эл. подпись/ signature)

Пенской
Александр
Владимирович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Бураков Илья Алексеевич

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42142

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Системное и прикладное программное обеспечение 2021

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Тема ВКР/ Thesis topic Совершенствование метода синтеза на основе машинного обучения в САПР специализированных вычислителей

Руководитель ВКР/ Thesis supervisor Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Ранее, в рамках ВКР бакалавра студентом разработан прототип, применяющий подходы из области машинного обучения (МО) к синтезу в САПР специализированных вычислителей. Необходимо усовершенствовать разработанное решение по обозначенным далее направлениям.

Цель работы: повысить эффективность САПР спецвычислителей за счет усовершенствования механизмов синтеза на базе машинного обучения.

Задачи:

- 1) Ревизия существующих механизмов синтеза на основе машинного обучения.
- 2) Разработка стратегии обхода дерева синтеза, поддерживающей откаты из тупиковых ветвей синтеза.
- 3) Проектирование и реализация нового метода сбора тренировочных данных.
- 4) Решение технических вопросов: программная интеграция с NITTA и контейнеризация рабочего окружения.

Содержание работы:

1. Введение. Актуальность темы. Постановка проблемы. Цели и задачи исследования.
2. Ревизия и анализ существующих механизмов синтеза на основе МО.

3. Проектирование и разработка новой стратегии обхода дерева синтеза, позволяющей возвращаться из тупиковых ветвей.
4. Реализация возможности сбора тренировочных данных с больших деревьев.
5. Программная интеграция синтеза на основе МО в поток исполнения САПР NITTA. Контейнеризация рабочего окружения.
6. Заключение. Выводы. Анализ возможных направлений для дальнейших исследований.

Исходные данные к работе: публичный репозиторий САПР спецвычислителей NITTA, рекомендуемые материалы и источники.

Рекомендуемые материалы:

1. Пенской А.В., Платунов А.Е., Ключев А.О., Горбачев Я.Г., Яналов Р.И. Система высокоуровневого синтеза на основе гибридной реконфигурируемой микроархитектуры, 2019.
2. Silver D. et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm // arXiv preprint arXiv:1712.01815. – 2017.

Форма представления материалов ВКР / Format(s) of thesis materials:

- 1) Презентация в формате PDF.
- 2) Текст ВКР в формате PDF.
- 3) Программная документация.

Дата выдачи задания / Assignment issued on: 23.09.2022

Срок представления готовой ВКР / Deadline for final edition of the thesis 30.05.2023

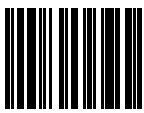
Характеристика темы ВКР / Description of thesis subject (topic)

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: да / yes

СОГЛАСОВАНО / AGREED:

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
11.05.2023	

(эл. подпись)

Пенской
Александр
Владимирович

Задание принял к
исполнению/ Objectives
assumed BY

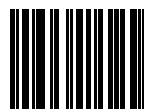
Документ подписан	
Бураков Илья Алексеевич	
11.05.2023	

(эл. подпись)

Бураков Илья
Алексеевич

Руководитель ОП/ Head
of educational program

Документ подписан
Бессмертный Игорь Александрович
02.06.2023



Бессмертный
Игорь
Александрович

(эл. подпись)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Бураков Илья Алексеевич
Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники
Группа/Group P42142
Направление подготовки/ Subject area 09.04.04 Программная инженерия
Образовательная программа / Educational program Системное и прикладное программное обеспечение 2021
Язык реализации ОП / Language of the educational program Русский
Статус ОП / Status of educational program
Квалификация/ Degree level Магистр
Тема ВКР/ Thesis topic Совершенствование метода синтеза на основе машинного обучения в САПР специализированных вычислителей
Руководитель ВКР/ Thesis supervisor Пенской Александр Владимирович, кандидат технических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Повысить эффективность САПР спецвычислителей за счет усовершенствования механизмов синтеза на базе машинного обучения (МО).

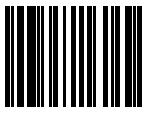
Задачи, решаемые в ВКР / Research tasks

1) Ревизия существующих механизмов синтеза на основе МО. 2) Разработка и испытание стратегии обхода дерева синтеза, поддерживающей откаты из тупиковых ветвей. 3) Разработка и испытание итеративного метода сбора тренировочных данных. 4) Решение технических вопросов: программная интеграция с NITTA и контейнеризация рабочего окружения.

Краткая характеристика полученных результатов / Short summary of results/findings

Новая стратегия обхода дерева синтеза значительно ускоряет синтез спецвычислителей в САПР NITTA, в особенности – для сложных целевых алгоритмов с большими деревьями синтеза. Новый итеративный метод сбора тренировочных данных позволяет повысить генерализуемость моделей МО, использующихся в качестве оценочной функции при синтезе спецвычислителей в САПР NITTA. Предложенные усовершенствования механизмов синтеза позволили повысить эффективность САПР спецвычислителей NITTA до 6 раз (зависит от сложности синтезируемых алгоритмов, наиболее выражена для сложных алгоритмов).

Обучающийся/Student

Документ подписан	
Бураков Илья Алексеевич	
04.06.2023	

(эл. подпись/ signature)

Бураков Илья
Алексеевич

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Пенской Александр Владимирович	
04.06.2023	

(эл. подпись/ signature)

Пенской
Александр
Владимирович

(Фамилия И.О./ name
and surname)

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	9
ВВЕДЕНИЕ	11
1 РЕВИЗИЯ СУЩЕСТВУЮЩИХ МЕХАНИЗМОВ СИНТЕЗА.....	21
1.1 Синтез в САПР NITTA.....	21
1.1.1 Структуры данных, применяемые в процессе синтеза	21
1.1.2 Обход дерева синтеза	22
1.2 Методы синтеза на основе машинного обучения	24
1.2.1 Применение модели машинного обучения к синтезу в САПР.....	25
1.2.2 Выявленные направления дальнейших исследований.....	26
1.3 Выводы	27
2 СТРАТЕГИЯ ОБХОДА ДЕРЕВА СИНТЕЗА.....	28
2.1 Ревизия существующих стратегий обхода дерева синтеза	28
2.1.1 Стратегии обхода в САПР NITTA.....	28
2.1.2 Анализ решений в других предметных областях.....	29
2.2 Проектирование и реализация новой стратегии обхода	33
2.3 Испытание разработанной стратегии обхода.....	36
2.4 Выводы	38
3 СБОР ТРЕНИРОВОЧНЫХ ДАННЫХ ИТЕРАТИВНЫМ МЕТОДОМ	39
3.1 Ревизия существующего метода сбора тренировочных данных	39
3.2 Проектирование итеративного метода сбора данных	42
3.2.1 Итеративность и стохастичность нового метода сбора данных.....	42
3.2.2 Проблема итеративной нормализации метрик листьев	43
3.2.3 Проблема коллизии путей и агрегирование данных	43
3.2.4 Предполагаемые недостатки итеративного метода сбора данных	44
3.3 Реализация итеративного метода сбора данных.....	44
3.3.1 Затухание тренировочных меток при обратном распространении.....	45
3.3.2 Параллелизация итеративного процесса сбора данных.....	46
3.3.3 Аппроксимация покрытия дерева синтеза при сборе данных	46
3.3.4 Улучшение качества кодовой базы	49
3.4 Испытание итеративного метода сбора данных.....	49
3.4.1 Ручное тестирование итеративного сбора данных.....	49

3.4.2	Тестирование производительности параллельного сбора данных	51
3.4.3	Сбор данных для подбора тренировочных целевых алгоритмов.....	52
3.4.4	Сбор тренировочных данных для обучения и испытания модели	53
3.5	Выводы	54
4	ВСТРАИВАНИЕ СИНТЕЗА НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ В NITTA	56
4.1	Исследование технических аспектов синтеза в NITTA	56
4.2	Программная интеграция прототипа и NITTA.....	57
4.2.1	Поиск способов интеграции прототипа и кода NITTA.....	57
4.2.2	Сравнительный анализ способов интеграции	58
4.2.3	Проектирование конечного решения.....	60
4.3	Реализация интеграции прототипа и процесса синтеза в NITTA	62
4.4	Тестирование, отладка и документирование реализации.....	66
4.5	Выводы	67
5	КОНТЕЙНЕРИЗАЦИЯ РАБОЧЕГО ОКРУЖЕНИЯ.....	68
5.1	Определение списка необходимых средств	68
5.2	Реализация и отладка сборочных скриптов.....	70
5.3	Организация поддержки графических процессоров	71
5.4	Интеграция решения в проект и его апробация.....	72
5.4.1	Подбор и проверка средств удаленной разработки в контейнере.....	72
5.4.2	Написание руководства по эксплуатации для разработчиков	73
5.4.3	Проверка работы контейнера на разных операционных системах.....	74
5.5	Выводы	74
	ЗАКЛЮЧЕНИЕ	76
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	77

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ВКР – выпускная квалификационная работа.

ВМ – виртуальная машина.

ИИ – искусственный интеллект.

МК – микроконтроллер.

МО – машинное обучение.

ОС – операционная система.

ПК – персональный компьютер.

ПЛИС – программируемая логическая интегральная схема.

ПО – программное обеспечение.

САПР – система автоматического проектирования.

ABI – Application Binary Interface – двоичный (бинарный) интерфейс, описывающий способы взаимодействия одного компонента компьютерной программы или с другим на уровне вызовов скомпилированных подпрограмм.

API – Application Programming Interface – интерфейс, описывающий способы взаимодействия одной компьютерной программы или компонента с другой программой или компонентом.

ASIC – Application-Specific Integrated Circuit – интегральная схема специального назначения.

CLI – Command-Line Interface – интерфейс командной строки.

CPU – Central Processing Unit – центральный процессор компьютера.

CSV – Comma-Separated Values – «значения, разделённые запятой» – текстовый формат сериализации табличных данных.

CUDA – Compute Unified Device Architecture – программная платформа, разработанная в компании NVIDIA, позволяющая использовать GPU для вычислений общего назначения.

DFG – Data-Flow Graph – граф пересылок данных в реализации САПР NITTA.

DX – Developer Experience – опыт взаимодействия разработчика с разрабатываемым ПО. Термин применяется в схожем контексте с UX (User experience, опыт пользователя).

FFI – Foreign Function Interface – механизм, благодаря которому программа, написанная на одном языке программирования, может делать вызовы к программам, написанным на другом языке.

GHC – Glasgow Haskell Compiler – компилятор для языка Haskell с открытым исходным кодом, разработанный в Университете Глазго.

GIL – Global Interpreter Lock – глобальная блокировка интерпретатора, особенность используемой реализации языка Python (CPython).

GPU – Graphics Processing Unit – специализированный компонент компьютера, предназначенный для обработки графических данных.

HDL – Hardware Description Language – специализированный язык для описания структуры и поведения интегральных схем.

HLS – High-Level Synthesis – синтез RTL-спецификации алгоритма, заданного на высокоуровневом языке, в САПР специализированных вычислителей.

HTTP – Hypertext Transfer Protocol – «протокол передачи гипертекста» – сетевой клиент-серверный протокол прикладного уровня.

IDE – Integrated Development Environment – интегрированная среда разработки.

IPC – Inter-Process Communication – межпроцессное взаимодействие.

MCTS – Monte Carlo Tree Search – метод Монте-Карло для поиска в графе.

RPC – Remote Procedure Call – вызов удаленных процедур.

RTL – Register-Transfer Level – уровень абстракции описания интегральной схемы, на котором алгоритм работы задаётся последовательностью операций над данными при их пересылке между регистрами.

SSH – Secure Shell – протокол, позволяющий защищённо подключаться к удаленным компьютерам.

UCT – «Upper Confidence Bounds Applied to Trees» – метод решения дилеммы о разведке и эксплуатации в контексте поиска в дереве методом Монте-Карло.

WSL – Windows Subsystem for Linux – встроенное в ОС Windows средство паравиртуализации для создания интегрированного окружения Linux.

ВВЕДЕНИЕ

Настоящая диссертация посвящена теме совершенствования метода синтеза на основе машинного обучения в системе автоматического проектирования (САПР) специализированных вычислителей. Чтобы сделать тему исследования понятнее, а также обосновать её актуальность и оценить степень разработанности, предлагается предварительно погрузиться в предметную область. Перед описанием проблемы, решаемой в настоящем исследовании, будет рассмотрено, что такое специализированные вычислители, где они применяются, как разрабатываются и какую роль профильные САПР могут играть в их разработке. Затем будет произведён обзор текущих проблем, сдерживающих развитие подобных САПР, а также объяснено, как машинное обучение (МО) применяется для решения некоторых из них.

Введение в предметную область. Прежде всего, рассмотрим понятие специализированного вычислителя. В современном мире цифровой электроники находят самые разные применения во многих областях человеческой деятельности. Часто производственные, научные и прочие процессы удаётся автоматизировать средствами общего назначения: применяются полноценные персональные компьютеры (ПК) с процессорами общего назначения (central processing unit, CPU) и соответствующим системным и прикладным программным обеспечением (ПО). Для решения некоторых задач удаётся задействовать и графические сопроцессоры (graphics processing unit, GPU), в том числе и для аппаратного ускорения неграфических вычислений. Там, где полноценные ПК неприменимы по причине дороговизны, массогабаритных характеристик, относительно высокого энергопотребления или отсутствия гарантий реального времени, могут быть задействованы микроконтроллеры (МК) – небольшие однокристальные компьютеры, подходящие для выполнения относительно простых задач. Как правило, они значительно дешевле (при сопоставимой серийности) и меньше ПК, однако всё ещё схемотехнически не специализированы под конкретную задачу, хоть и стоит отметить возможность наличия особенностей конфигурации (например, присутствие интерфейсов ввода-вывода конкретного стандарта), которые могут делать некоторые модели удобнее других при применении в заданных обстоятельствах.

Указанные средства общего назначения имеют как свои преимущества, так и недостатки, которые учитывают при выборе решения прикладной задачи. Эти особенности должны быть согласованы с требованиями, диктуемыми спецификой автоматизируемых процессов. По этой причине для некоторых задач обозначенные ранее вычислители общего

назначения не подходят [1]. В подобных случаях могут быть применены *специализированные вычислители* – вычислительные устройства, спроектированные для решения одной или нескольких конкретных задач. Они обладают рядом уникальных преимуществ: например, могут обеспечить детерминированность времени работы алгоритма, высочайшую скорость обработки данных и исключительно низкое энергопотребление. Спецвычислители находят применение в следующих областях:

- обработка сигналов (например, аудио и видео: кодирование и декодирование, фильтрация, распознавание изображений и речи в реальном времени) [1, 2];
- киберфизические системы (автоэлектроника, аэрокосмическая и оборонная индустрии, автоматизированные системы управления технологическим процессом – АСУ ТП, сенсоры, регуляторы, системы отслеживания объектов, интернет вещей) [3–6];
- машинное обучение [5–7];
- медицинская электроника и биоинформатика [6];
- финансовые технологии (высокочастотная торговля, аппаратное ускорение добычи криптовалют) [8, 9];
- моделирование (в частности, сложные физические симуляции и вычислительная химия, а также моделирование в реальном времени) [10, 11];
- другие индустрии и области человеческой деятельности [1, 6].

В то время как существуют различные по принципу функционирования виды спецвычислителей, настоящая работа актуальна прежде всего для цифровых. Они могут быть реализованы как непосредственно в виде интегральных схем специального назначения (*application-specific integrated circuit, ASIC*), так и на базе программируемых логических интегральных схем (ПЛИС). ASIC отличаются тем, что их топология зафиксирована после создания. Это позволяет не только удешевить производство больших партий интегральных схем, но и получить наилучшее в классе быстродействие и энергопотребление за счёт сокращения накладных расходов вычислительного процесса, присущих ПЛИС. ПЛИС же, в свою очередь, обладают другим преимуществом – реконфигурируемостью, то есть возможностью реализовывать на одном и том же физическом чипе различные логические схемы, а также менять их по запросу пользователя благодаря особой ячеечной архитектуре и программируемым соединениям между логическими элементами и вычислительными блоками. Такая особенность полезна при быстром прототипировании, производстве небольшой серии решений с заданной логической схемой, а также в случаях, где можно получить преимущество за счёт

динамической реконфигурируемости, то есть способности ПЛИС изменять конфигурацию непосредственно во время эксплуатации.

Чтобы разработать специализированный вычислитель, необходимо, как правило, создать описание интегральной схемы на уровне регистровых передач (Register-Transfer Level, RTL) с использованием специализированных языков (Hardware Description Language, HDL) [12]. Примерами популярных HDL могут послужить Verilog и VHDL. Из данного представления с помощью средств логического синтеза может быть сгенерировано описание реализации интегральной схемы на уровне логических вентилях, преобразуемое затем в топологию ASIC или прошивку для ПЛИС.

Написание и поддержка RTL-спецификаций вручную требует привлечения специалистов узкого профиля и высоких трудозатрат, особенно в ситуациях, когда в прикладной алгоритм часто вносятся корректировки [13]. Альтернативой является генерация RTL-спецификации средствами систем высокоуровневого синтеза (High-Level Synthesis, HLS) или применение САПР специализированных вычислителей, позволяющих использовать языки высокого уровня для описания поведения синтезируемого прикладного алгоритма (далее называется *целевым алгоритмом*). В таком случае «RTL-спецификация является результатом работы инструментальных средств и представляет собой совокупность описания цифровых схем и (при необходимости) программного обеспечения» для их компонентов [13].

Актуальность и степень разработанности темы. Тема высокоуровневого синтеза спецвычислителей с помощью профильных САПР является достаточно проработанной научным и профессиональным сообществом [14–16]. Идея HLS исследуется в академической среде на уровне прототипов с 1970-х, а в 90-х появились ранние коммерческие средства, которые укрепили основы подхода и привлекли широкое внимание, хоть и не заменили рабочие процессы на RTL в сообществе разработчиков ASIC и ПЛИС [16]. Тем не менее, потребность в подобной инструментарии высокого качества не угасала. Появлялось всё больше факторов, вынуждающих повышать уровень абстракции над RTL (прежде всего, растущая сложность интегральных схем). Существующие наработки и средства совершенствовались.

В 2011 появились доступные, достаточно развитые и получившие признание в индустрии САПР, такие как AutoPilot от AutoESL. В некоторых предметных областях при сравнении одних спецвычислителей, сгенерированных средствами HLS, и других спецвычислителей, вручную разработанных экспертами на уровне RTL, результат работы средств HLS был по производительности и стоимости спецвычислителей либо похож, либо лучше ручного [16], не говоря о стоимости времени разработчиков. AutoPilot на данный

момент продолжает существовать (переименовано в Vivado HLS, а затем в Vitis HLS, разработчик – компания Xilinx, что с 2022 принадлежит AMD) и развиваться [15], как и многие другие САПР, например:

- Intel HLS Compiler [17] (коммерческая);
- Catapult HLS [18] (коммерческая);
- PandA Bambu HLS Framework [19] (открытая);
- LegUp (создавалась как открытая [20], но в 2020 была куплена компанией Microchip, переименована в SmartHLS и коммерциализована [21], поддержка ПЛИС сторонних производителей вырезана, а исходный код закрыт);

Однако существующие средства HLS обладают рядом ограничений, которые, главным образом, заключаются «в необходимости глубокого понимания схемотехники для эффективного использования, высокой сложности, непрозрачности, специализированности под конкретные виды задач» [13]. Отдельно стоит отметить склонность коммерческих САПР с закрытым исходным кодом иметь строгое и/или дорогостоящее лицензирование, быть негибкими к расширению и модификации, а также вынуждать использовать оборудование конкретных поставщиков [12]. В свою очередь, открытые САПР часто уступают в проработанности и качестве результатов коммерческим САПР ввиду исключительной сложности последних [15].

Таким образом, разработка и усовершенствование САПР спецвычислителей с открытым исходным кодом – актуальная и практически значимая тема исследований. В настоящий момент существуют и разрабатываются подобные САПР, являющиеся попыткой решить обозначенные проблемы [12]. Среди них – САПР NITTA, на примере которой производится реализация и тестирование решений в настоящей работе.

Решаемая проблема. Разработка САПР спецвычислителей сопряжена со многими проблемами. В настоящем исследовании решается одна из них – комбинаторная сложность задачи синтеза в условиях ограниченного времени. Чтобы понять, в чём именно сложность, предлагается рассмотреть внутренние процессы в САПР подробнее.

Процесс синтеза – процесс преобразования описания целевого алгоритма (желаемого поведения спецвычислителя) на высокоуровневом языке в соответствующую *модель вычислителя* – внутреннее представление спецвычислителя в САПР, содержащее информацию о структуре вычислителя. Из полученной в качестве результата синтеза модели вычислителя будет сгенерирована RTL-спецификация, преобразуемая далее в описание на уровне логических вентилях, а затем в прошивку для ПЛИС или топологию ASIC.

Процесс синтеза для получения результата должен решить ряд задач. Во-первых, должна быть решена задача о структуре спецвычислителя. В системе NITTA используется оригинальная архитектура [22], предполагающая синтез структуры спецвычислителей из неэлементарных компонентов, чьи функционал и реализация на HDL определены заранее (сумматоры, регистры, интерфейсы ввода-вывода, и так далее). Эти компоненты названы *вычислительными блоками*. При синтезе должен быть определён используемый набор этих вычислительных блоков, то есть необходимое количество блоков каждого типа, а также конфигурация соединений между ними (количество и расположение шин). Полученная спецификация структуры спецвычислителя названа *микроархитектурой*.

Во-вторых, должна быть решена задача о поведении спецвычислителя, то есть синтезирована управляющая программа, реализующая циклический вычислительный процесс, соответствующий целевому алгоритму. Управляющая программа задаётся в виде расписания, определяющего, чем занят каждый конкретный вычислительный блок в микроархитектуре на каждом конкретном такте в цикле вычислительного процесса.

Для решения описанных задач синтеза САПР NITTA работает со структурой данных, называемой *деревом синтеза*, которое схематично изображено на рисунке 1.

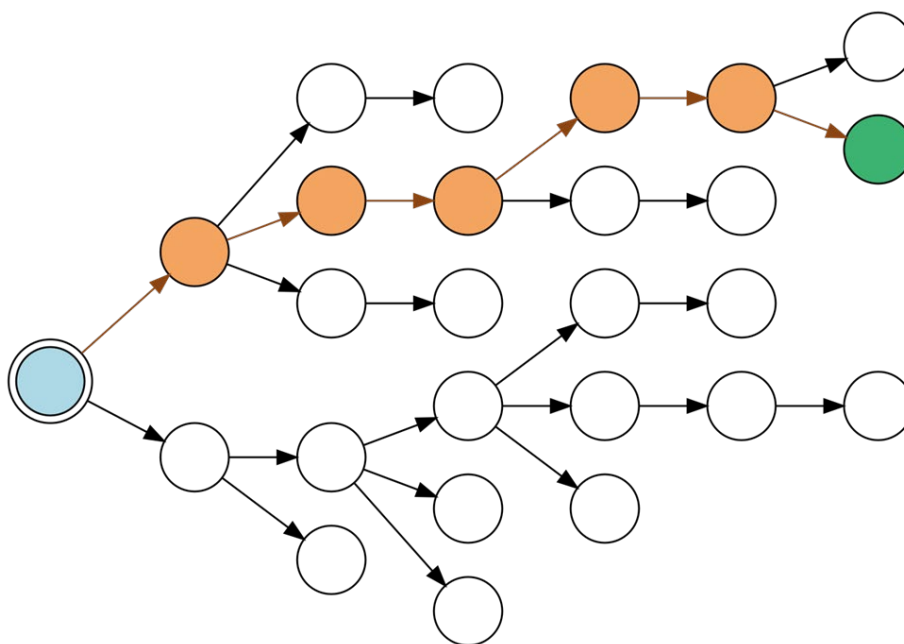


Рисунок 1 – Дерево синтеза и путь в нём

Каждая вершина в этом дереве – состояние модели вычислителя, а каждое ребро – переход в соответствии с принятым решением по изменению этого состояния модели вычислителя. Например, «добавить ещё один регистр на шину 4 в микроархитектуру», «запланировать пересылку данных в такте №10» или «назначить функцию сумматору X».

Рёбра соответствуют подобным действиям, которые изменяют состояние модели вычислителя, поэтому каждое ребро ведёт от исходного состояния к изменённому.

При такой постановке вопроса процесс синтеза представляет собой обход данного дерева. Он начинается в корне (обозначен голубым на рисунке) и движется в сторону листьев. В каждой вершине имеется множество решений синтеза, которые можно принять, то есть множество исходящих рёбер, по которым можно продолжать спуск по дереву синтеза. Одна из возможных цепочек решений обозначена коричневым.

Когда движение завершается в листе дерева, мы получаем результат синтеза. Он может быть неуспешным (красные листья на рисунке 2) или успешным (зелёные листья). Критерий успешности будет рассмотрен при более подробном описании процесса синтеза в разделе 1. В случае неуспешного результата обход дерева синтеза можно продолжить из других вершин.

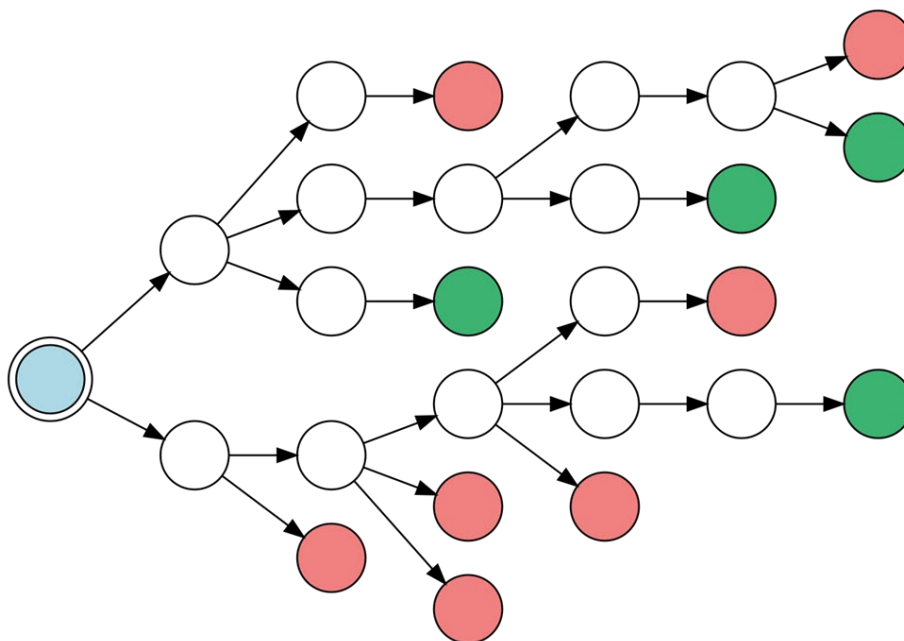


Рисунок 2 – Листья дерева синтеза

Основная проблема реализации процесса синтеза – мощность дерева синтеза может быть слишком велика. Происходит «комбинаторный взрыв», который делает полный обход дерева синтеза на практике недостижимым [13, 23]. Например, дерево синтеза такого относительно небольшого устройства, как пропорционально-интегрально-дифференцирующий регулятор, может уже состоять из десятков миллионов вершин. Требуются решения эвристического характера, предварительно оценивающие вершины перед вычислением их детей и исключаяющие из рассмотрения наименее перспективные, сокращая таким образом мощность фактически обходимого поддерева и значительно ускоряя процесс синтеза.

Существенная работа в указанном направлении проведена как другими разработчиками САПР NITTA (вручную составленные эвристики на основе экспертных знаний), так и в рамках более ранних трудов автора, прежде всего – выпускной квалификационной работы (ВКР) бакалавра [13]. Автором разработан прототип метода синтеза, применяющий подходы из области машинного обучения (МО) к решению обсуждаемой проблемы, то есть к задаче поиска оптимального результата в комбинаторно больших деревьях синтеза с учётом ограниченных вычислительных ресурсов. Критерий оптимальности при этом может задаваться на усмотрение разработчика САПР при применении прототипа. Модель МО в прототипе применялась для оценки перспективности исследования поддерева каждой конкретной вершины и, таким образом, управляла тем, какие поддерева останутся без рассмотрения в процессе синтеза.

Работа в выявленных ранее направлениях дальнейших исследований позволит усовершенствовать прототип и приблизить решение к применимому на практике средству, чему и посвящено настоящее исследование.

Среди возможных улучшений – реализация возможности сбора тренировочных данных с больших, потенциально бесконечных деревьев синтеза. В старой версии прототипа было необходимо полностью вычислить дерево синтеза, и только после этого его можно было преобразовать в экземпляры данных для обучения модели. Данный метод не подходит для преобразования деревьев, не помещающихся в оперативную память. Обойти это ограничение можно с помощью нового итеративного метода сбора данных, который предлагается спроектировать и реализовать. Предстоит организовать сбор данных итеративно, обрабатывая по одному случайному пути в дереве синтеза за итерацию. Такой подход позволит высвободить дерево из оперативной памяти, когда она закончится, и продолжать осуществлять сбор данных необходимое количество итераций до достижения желаемого собранного объёма.

Ожидается, что новый подход позволит значительно увеличить количество и репрезентативность собранных данных для обучения, а это, в свою очередь, положительно скажется на *генерализуемости модели* МО по сравнению с разработанной ранее версией прототипа. Генерализуемостью модели называется способность сохранять предсказательную силу на эксплуатационных данных, а не только на тренировочном наборе. Например, при применении модели на деревьях синтеза новых целевых алгоритмов. Измерить генерализуемость моделей сравниваемых версий прототипа можно будет, оценив способность САПР успешно синтезировать новые целевые алгоритмы с деревьями синтеза большого объёма (миллион вершин и более) при применении метода синтеза на основе МО с различными сравниваемыми моделями.

Кроме того, выявлено и другое возможное усовершенствование прототипа. В оригинальной версии модель МО не могла спровоцировать *откат* процесса синтеза наверх по дереву из бесперспективных ветвей, даже если в какой-то момент оценки модели начинали говорить о том, что это было бы разумно. Откатом называется возврат по дереву синтеза в посещённую ранее вершину, чтобы продолжить дальнейший спуск из неё. Он может быть целесообразен, например, если становится известно, что дальнейший спуск в вершин-детей из текущей позиции заведомо не приведёт к успешному результату синтеза. Невозможность откатов можно исправить, разработав новую *стратегию обхода* дерева синтеза – набор правил, определяющих, в каком порядке будет производиться обход вершин в дереве синтеза. Реализация такой возможности, предположительно, позволит значительно ускорить синтез в случаях, когда старая стратегия натывается на неудачные результаты синтеза, что часто происходит в больших деревьях синтеза относительно сложных целевых алгоритмов.

Научная новизна. Идея применить МО для ускорения поиска в проектировочном пространстве не является новой сама по себе. Подобные исследования уже существуют, в том числе в контексте HLS [24–29], некоторые из них опубликованы в рецензируемых изданиях непосредственно во время написания диссертации, что является ещё одним доказательством актуальности темы, а результаты этих исследований служат дополнительным подтверждением применимости подхода. В то же время, решаемая в настоящей работе проблема часто в деталях отличается от поставленной в других исследованиях. Например, в [24] средство HLS считается внешним, законченным продуктом, и производится поиск только среди возможных «подкруток» (в оригинале – «knobs»), подсказывающих выбранной САПР, как именно синтезировать ту или иную часть поведения целевого алгоритма. Другими словами, исследование Н.-У. Liu и L. Carloni не относится к разработке САПР спецвычислителей и не ставит целью решить обозначенные ранее проблемы существующих средств HLS.

Соответственно, новым как в ВКР бакалавра автора, так и настоящей работе, является применение МО именно к реализации синтеза внутри самой САПР. Такой подход не удалось найти в других опубликованных исследованиях. Настоящая диссертация, в свою очередь, является непосредственным развитием и логическим продолжением ВКР бакалавра автора [13]. Прорабатываются вопросы, обозначенные в [13] как «направления дальнейших исследований», что подтверждает научную новизну настоящей работы в отдельности.

Цель исследования. Повысить *эффективность САПР спецвычислителей* за счет усовершенствования механизмов синтеза на базе машинного обучения (МО). Критериями

эффективности САПР при этом, как и в ВКР бакалавра, считаются длительность процесса синтеза и длительность полученной управляющей программы в тактах. Чем меньше значения упомянутых величин, тем выше считается эффективность САПР. Кроме того, второй критерий относится к зависимым от результатов синтеза и, в общем случае, может быть заменён при применении прототипа на усмотрение разработчика (например, если есть желание минимизировать количество используемых вычислительных блоков за счёт увеличения длительности управляющей программы).

Задачи исследования. Обобщая предлагаемые усовершенствования, можно сформулировать задачи, решаемые в настоящей работе:

- 1) ревизия существующих механизмов синтеза на основе МО;
- 2) разработка и испытание стратегии обхода дерева синтеза, поддерживающей откаты из тупиковых ветвей;
- 3) разработка и испытание итеративного метода сбора тренировочных данных;
- 4) решение технических вопросов: программная интеграция с NITTA и контейнеризация рабочего окружения.

Начать работу видится разумным с ревизии существующих механизмов синтеза на основе МО, что позволит описать все необходимые связанные понятия, а также принимать более обоснованные проектировочные решения на дальнейших этапах.

Также необходимо решить ряд технических вопросов, связанных с программными аспектами реализации и испытания обсуждаемых усовершенствований. В них входит интеграция метода синтеза на основе МО с САПР NITTA на программном уровне, а также контейнеризация рабочего окружения для автоматизации его настройки, что упростит как воспроизведение проведённых в рамках исследования испытаний, так и использование САПР с прототипом для будущих разработчиков и пользователей.

Методы исследования. Анализ и синтез, дедукция, индукция, метод аналогий, сравнение и эксперимент.

Положения, выносимые на защиту:

- 1) Новая стратегия обхода дерева синтеза значительно ускоряет синтез спецвычислителей в САПР NITTA, в особенности – для сложных целевых алгоритмов с большими деревьями синтеза.
- 2) Новый итеративный метод сбора тренировочных данных позволяет повысить генерализуемость моделей МО, использующихся в качестве оценочной функции при синтезе спецвычислителей в САПР NITTA.

Практическая значимость работы. Работа является практически значимой, поскольку успешно усовершенствовала процесс синтеза (ускорила, улучшила качество

результатов в соответствии с установленными критериями) в бесплатной САПР спецвычислителей NITTA с открытым исходным кодом и разрешительной лицензией BSD. В свою очередь, практическая значимость как САПР спецвычислителей (прежде всего, открытых), так и спецвычислителей в целом обосновывается широким спектром возможных применений в различных областях человеческой деятельности и более подробно рассмотрена ранее.

Степень достоверности и апробация результатов. Результаты исследования были апробированы посредством их рассмотрения научным руководителем и рецензентом, а также их неформальным обсуждением с квалифицированными членами профессионального сообщества. Выводы о результатах основаны на эмпирических данных – измерениях, полученных в ходе проведённых экспериментов. Исходный код, разработанный для проведения экспериментов, опубликован в открытый доступ добавлением в репозиторий САПР NITTA [30], что позволяет независимо воспроизвести полученные результаты. Таким образом, степень достоверности результатов можно считать достаточной.

Рассмотрены все основные вопросы, требовавшие внимания во введении. Предлагается перейти к ревизии существующих механизмов синтеза, чтобы затем приступить к проектированию, реализации и испытанию предлагаемых усовершенствований.

1 РЕВИЗИЯ СУЩЕСТВУЮЩИХ МЕХАНИЗМОВ СИНТЕЗА

В настоящем разделе описывается реализация синтеза в САПР NITTA. Рассмотрены используемые структуры данных и процедура обхода дерева синтеза. Показано, как автором в его предшествующих работах было применено МО для ускорения процесса синтеза и создан упомянутый во введении прототип метода синтеза на основе МО [13]. Производится обзор реализации прототипа, описываются выявленные направления дальнейших исследований.

1.1 Синтез в САПР NITTA

1.1.1 Структуры данных, применяемые в процессе синтеза

Процесс синтеза в NITTA может быть глобально описан следующим образом. На вход процессу, если опустить технические детали, подаётся исходный код целевого алгоритма на подмножестве высокоуровневых языков. На момент написания (май 2023) поддерживаются языки Lua и XMILE, но предусмотрены программные абстракции для упрощения расширения этого списка в будущем.

Исходный код преобразуется в граф пересылок данных (Data-Flow Graph, DFG), который не стоит путать с деревом синтеза. DFG отражает сам вычислительный процесс в виде ориентированного графа, в котором вершины – функции вычислительных блоков, а рёбра – пересылки значений между ними. Эта информация выделяется из операторов и вызовов функций высокоуровневого языка. Теперь задачи, которые нужно решить процессу синтеза, можно переформулировать более подробно:

- 1) найти микроархитектуру (количество необходимых вычислительных блоков каждого типа), способную реализовать синтезируемый вычислительный процесс;
- 2) назначить все заданные функции конкретным вычислительным блокам;
- 3) потактово запланировать все требуемые пересылки данных;
- 4) осуществить все необходимые *эквивалентные преобразования DFG*, то есть преобразования, не нарушающие смыслового соответствия целевого алгоритма и получаемого DFG.

Из этих «низкоуровневых» задач прямо следуют виды действий, которые можно сделать с моделью спецвычислителя в процессе синтеза:

- *добавление блока* (allocation decision): добавить необходимый вычислительный блок в микроархитектуру;
- *назначение функции* (binding decision): запланировать исполнение функции конкретным вычислительным блоком;
- *пересылка данных* (dataflow decision): запланировать на конкретный такт передачу какого-либо значения по шине между вычислительными блоками;
- *преобразование* (refactoring decision): совершить эквивалентное преобразование DFG (понятие определено выше).

Более подробно эти виды действий описаны в [13]. Они соответствуют видам рёбер, которые могут существовать в дереве синтеза. Что касается его вершин, то в них, как уже отмечено, находятся состояние модели спецвычислителя, которое эти действия могут изменять.

Определим, чем задаётся состояние модели спецвычислителя. Для этого введём понятие *сети на основе шины* (bus network). Это объединение вычислительных блоков, расположенных на одной шине и контролируемых одной управляющей программой, вместе со связанной с ними информацией о запланированных пересылках данных и назначенных блокам функциях, помимо прочего. На данный момент NITTA поддерживает только наличие одной такой сети в синтезируемой микроархитектуре. С учётом этого, в состояние модели спецвычислителя входят:

- текущее состояние микроархитектуры в виде состояния сети на основе шины (изменяется при добавлении вычислительных блоков, назначении функции блоку или планировке пересылки, в исходном коде NITTA – `TargetSystem.mUnit :: BusNetwork`);
- текущее состояние DFG (изменяется при преобразованиях, в коде – `TargetSystem.mDataFlowGraph :: DataFlowGraph`).

Итак, рассмотрены основные структуры данных, связанные с процессом синтеза, что позволяет далее описать, как протекает сам процесс и как он работает с деревом синтеза.

1.1.2 Обход дерева синтеза

Процесс начинается в корне дерева и пошагово движется в сторону листьев. Иллюстрация приведена на рисунке 3, один из возможных путей спуска обозначен коричневым. Понятие оценочной функции будет пояснено в ближайших абзацах. При

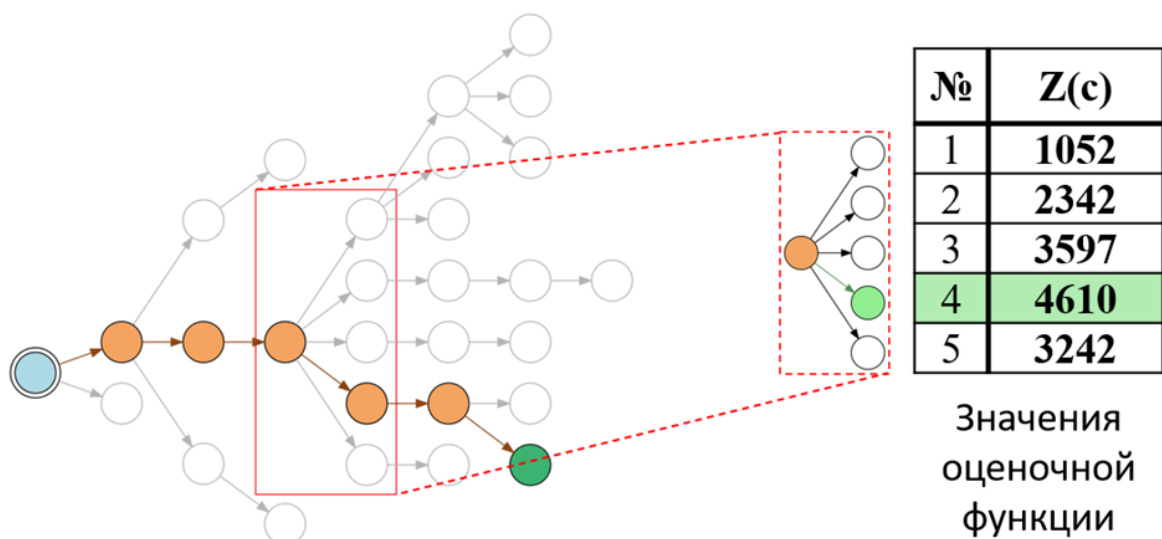


Рисунок 3 – Шаг процесса синтеза и применение оценочной функции

каждом шаге (пример шага на иллюстрации выделен красной рамкой) необходимо сделать выбор, в какую вершину (другими словами, по какому ребру) двигаться дальше. Выбранная вершина будет *посещена*, то есть будут вычислены её дети и сделан следующий шаг уже из неё.

На практике в связи с большой мощностью деревьев синтеза есть возможность за разумное время посетить лишь относительно небольшую часть вершин крупного дерева, поэтому необходимо при каждом шаге выбрать для посещения лишь наиболее *перспективные* вершины. Под перспективностью вершины (не стоит путать с успешностью результатов синтеза в листе) здесь понимается ценность исследования поддерева вершины с точки зрения получения желаемого результата синтеза в конце спуска. Желаемым может быть успешный результат синтеза с, например, как можно меньшей длительностью управляющей программы в тактах.

Для решения данной проблемы выбора разработчиками САПР NITTA ранее реализованы упомянутые во введении решения эвристического характера. Выбор осуществляется с помощью вводимой *оценочной функции*, которая призвана численно оценить состояние модели вычислителя в вершине на предмет перспективности. Оценка производится на основании различных *характеристик* как самого действия, применяемого в ребре, так и получаемой в результате действия модели спецвычислителя. Более подробно характеристики описаны в ВКР бакалавра автора [13], примерами могут послужить:

- тип действия,
- количество назначенных функций,
- длительность действия в тактах,

- количество альтернативных действий типа «связывание»,
- возможность появления взаимоблокировки в дальнейшем в результате действия.

При каждом шаге вычисляется значение оценочной функции для каждой вершины-кандидата. Посещается та вершина, чьё значение оценочной функции максимально (на рисунке 3 такой является вершина №4, выделена зелёным). Когда процесс оказывается в листе, то есть больше не осталось корректных возможных действий, то состояние модели спецвычислителя в этом листе является потенциальным результатом процесса синтеза.

Если все выявленные функции из DFG в данном состоянии назначены конкретным вычислительным блокам-исполнителям, а все необходимые пересылки данных успешно запланированы, можно говорить о получении успешного результата синтеза. Он далее может быть преобразован в HDL-описание спецвычислителя и ПО для его компонентов, что после логического синтеза будет представлять собой готовую к производству или прошивке на ПЛИС реализацию спецвычислителя.

Когда обозначенное выше условие успешности результата синтеза в листе не выполнено, полученная модель спецвычислителя некорректна и будет являться неуспешным результатом синтеза. Обход дерева может быть продолжен до получения успешного результата в соответствии со стратегией обхода, подробное рассмотрение которой предлагается отложить до раздела 2, посвящённого её усовершенствованию.

Таким образом, сформировано представление об основных понятиях и механизмах процесса синтеза. В следующем подразделе производится ревизия реализации прототипа, созданного автором в рамках ВКР бакалавра [13] и применяющего МО для ускорения описанного процесса синтеза, обсуждаются возможные направления дальнейших исследований.

1.2 Методы синтеза на основе машинного обучения

Первая оценочная функция в САПР NITTA, полученная без МО, – результат ручного эмпирического подбора методом проб и ошибок с применением экспертных знаний из предметной области. В то время как такая функция позволяла успешно синтезировать многие целевые алгоритмы, актуальным оставалось исследование вопроса о том, могут ли быть найдены более успешные оценочные функции. Другим мотивом дальнейших исследований было желание каким-либо образом автоматизировать процесс получения таких функций, чтобы было в долгосрочной перспективе проще поддерживать их релевантность меняющимся с течением времени особенностям реализации САПР NITTA.

В этом свете особый интерес вызывали индуктивные алгоритмы МО, которые математически «выявляют общие для какой-либо предметной области закономерности на основе известного конечного набора эмпирических данных» [23].

Исследование упомянутых вопросов было проведено в рамках ВКР бакалавра автора. Разработан метод синтеза, в котором оценочная функция является результатом обучения модели МО и, таким образом, может быть сгенерирована автоматически без применения каких-либо заданных эвристик из предметной области, а также обладает статистической обоснованностью.

Далее кратко рассмотрено, как именно машинное обучение было применено для решения поставленной проблемы, а также какие возможные направления дальнейших исследований были предложены. Проработка некоторых направлений станет содержанием дальнейших разделов.

1.2.1 Применение модели машинного обучения к синтезу в САПР

Модель МО в прототипе была предназначена для предварительной оценки вершин дерева синтеза без их посещения, то есть без вычисления их детей. Она используется в качестве оценочной функции в процессе синтеза, то есть применяется к каждой оцениваемой вершине-кандидату по отдельности и в качестве выходных данных возвращает численную оценку этой вершины. Иллюстрация процесса приведена на рисунке 4. Таким образом, с точки зрения МО решается задача регрессии.

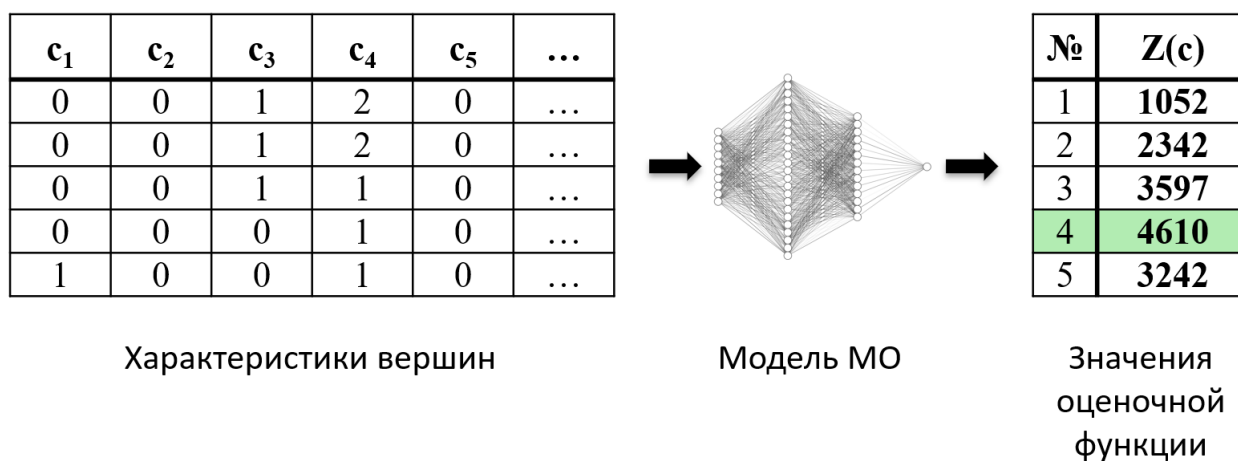


Рисунок 4 – Роль модели МО в процессе синтеза

Обучение производится с учителем (в англоязычной литературе – supervised learning [31]), то есть на основе заранее собираемого с деревьев синтеза целевых алгоритмов-

примеров набора тренировочных данных, экземпляры в котором состоят из пар «входные данные, метка». Меткой в контексте МО (label, target [31]) можно назвать заранее известный «правильный ответ» на соответствующие входные данные. Успешно обученная модель при подаче на неё тренировочных входных данных должна давать ответ, похожий на их метку. Сгенерированный набор данных с известными метками для обучения модели на основе деревьев синтеза целевых алгоритмов-примеров есть результат работы метода сбора данных, который станет одним из совершенствуемых аспектов обсуждаемого прототипа в следующих разделах.

Входными данными для применяемой модели МО (как и для любой оценочной функции) являются описанные в подразделе 1.1.2 характеристики оцениваемой вершины дерева. Некоторые характеристики для подачи на вход модели требуют определённых преобразований, в частности – приведения к числовому виду. Например, это необходимо для характеристик категориального типа, то есть типа с дискретным конечным множеством возможных несравнимых друг с другом значений (пример – характеристика «тип действия»). При сборе тренировочных данных они должны быть вычислены в заданном виде и записаны на диск вместе с меткой.

Среди достигнутых в ВКР бакалавра результатов отмечено, что «предложенный метод позволяет получить статистически обоснованную целевую функцию, с помощью которой реализуется эффективный алгоритм синтеза» [13]. Выводы сделаны на основании проведённого исследования эффективности. Таким образом, идея применить методы машинного обучения к реализации метода синтеза в САПР специализированных вычислителей оправдала себя. Тем не менее, отмечена необходимость дальнейших исследований в рамках данной тематики, которые позволят полностью раскрыть потенциал подхода.

1.2.2 Выявленные направления дальнейших исследований

Рекомендовалось, прежде всего, исследовать последствия значительного увеличения объёма тренировочных данных. Предсказательная сила моделей машинного обучения тем больше, чем ближе эксплуатационные распределения элементов данных к тренировочным. Следовательно, большее количество разнообразных тренировочных экземпляров с разных деревьев синтеза позволит получить модель, лучше обобщающуюся на реальные целевые алгоритмы. Увеличить объём данных для обучения позволит написание большего количества примеров целевых алгоритмов для сбора данных с их деревьев синтеза, ускорение выгрузки данных из САПР и их обработки, параллелизация

процесса сбора данных. Кроме того, значительно влияющим на объём тренировочных данных ограничением существующего метода является невозможность собирать данные с деревьев синтеза, не помещающихся в оперативную память. Данное направление было проработано в настоящей диссертации. Разработка усовершенствованного метода сбора тренировочных данных рассмотрена в разделе 3.

Кроме того, при исследовании эффективности прототипа в рамках ВКР бакалавра был обнаружен недостаток используемой тестовой стратегии обхода дерева: если процесс оказывался в листе с неудачным результатом синтеза, то классический поиск в глубину в дереве синтеза, в который вырождался процесс обхода, часто приводил к большому избыточному количеству вызовов оценочной функции и, соответственно, неудовлетворительному по эффективности САПР результату. Другие реализованные в САПР стратегии обхода позволяют обойти этот недостаток за счёт избыточного вычисления различных ветвей, но такой подход также является вычислительно дорогостоящим.

Такая ситуация обратила внимание на важный недостаток совершенствуемого прототипа: модель МО никак не может провоцировать откат из тупиковых или бесперспективных ветвей, когда оценки модели становятся низкими. Раздел 2 посвящён разработке новой стратегии обхода дерева синтеза, позволяющей это исправить.

Также предложено провести исследования применимости других моделей МО в данном формате: нейронных сетей с изменённой или принципиально другой архитектурой, моделей с байесовским подходом, случайного леса, других классических алгоритмов. Отмечена потенциальная уместность дополнительной инженерии признаков или расширения входных данных модели дополнительной информацией. Эти направления, хоть и сочтены актуальными, в настоящей работе исключены из рассмотрения как менее приоритетные, поскольку предполагается, что соответствующие изменения повлияют на эффективность САПР меньше отмеченных ранее.

1.3 Выводы

Произведён обзор реализации синтеза в САПР NITTA, описаны необходимые далее понятия используемых механизмов и структур данных. Рассмотрено, как МО ранее было применено для ускорения синтеза в САПР спецвычислителей. Описаны актуальные направления дальнейших исследований, подлежащие проработке в следующих разделах.

2 СТРАТЕГИЯ ОБХОДА ДЕРЕВА СИНТЕЗА

В рамках настоящего раздела решается задача разработки новой стратегии обхода дерева синтеза, поддерживающей откаты из тупиковых ветвей. Такое свойство необходимо для повышения эффективности САПР при её применении к сложным целевым алгоритмам, в деревьях синтеза которых часто встречаются листья с неуспешными результатами синтеза.

Рассмотрены существующие стратегии обхода, реализованные в САПР NITTA, а также проанализированы возможные решения проблемы из других предметных областей. Спроектирована, реализована и испытана оригинальная стратегия обхода дерева синтеза, поддерживающая откаты.

2.1 Ревизия существующих стратегий обхода дерева синтеза

2.1.1 Стратегии обхода в САПР NITTA

В САПР NITTA реализовано несколько стратегий обхода. Почти все они специфичны для предметной области и составлены вручную. Далее представлено их краткое описание с именованием, как у соответствующих им функций в кодовой базе.

SimpleSynthesisIO осуществляет все *очевидные назначения функций*, а затем находит лучший лист полным перебором. Очевидным назначением функции называется такое действие с моделью спецвычислителя вида «назначение функции», у которого нет возможных альтернативных назначений.

SmartBindSynthesisIO применяет все действия вида «преобразование DFG», пока они есть, и осуществляет все назначения функций с наилучшим значением оценочной функции. Когда не остаётся преобразований и назначений, находит лучший лист полным перебором.

BestThreadIO спускается к листу, выбирая следующую вершину по принципу максимума значения оценочной функции.

BestThreadIO + AllBindsAndRefsIO сначала спускается по всем действиям вида «назначение функции» или «преобразование DFG» с минимальными значениями оценочной функции, пока они есть, а затем действует согласно BestThreadIO.

StateOfTheArtSynthesisIO – находит четыре результата синтеза, по одному от каждой из упомянутых выше стратегий, и выбирает из них лучший на основе заданных критериев

(длительность управляющей программы, количество используемых вычислительных блоков).

При тестировании прототипа метода синтеза на основе МО в ВКР бакалавра автором использовалась тестовая стратегия обхода, ведущая себя похожим на BestThreadIO образом, но осуществляющая полный обход в глубину, если результат синтеза в листе оказался неуспешным.

Наибольший интерес представляет StateOfTheArtSynthesisIO. В испытаниях в качестве метода синтеза без МО используется эта стратегия обхода со стандартной оценочной функцией. В то же время, ни одна из упомянутых стратегий не поддерживает откаты как таковые, что подтверждает необходимость реализации новой стратегии обхода.

2.1.2 Анализ решений в других предметных областях

Перед проектированием собственного решения поставленной задачи произведён поиск и анализ существующего исследовательского опыта. К сожалению, не найдены публикации в предметной области диссертации (HLS), решающие проблемы с достаточно похожей постановкой задачи. Тем не менее, подходы к решению аналогичных задач обширно исследованы в некоторых других предметных областях.

Автором уже отмечалось в предыдущих работах сходство задач разработки синтеза в САПР спецвычислителей и искусственного интеллекта (ИИ) для настольных игр с доской (например, шахмат, го) [13]. В предметной области игрового ИИ присутствуют деревья игр, аналогичные деревьям синтеза: состояние игры в вершинах соответствует состоянию модели спецвычислителя, а ходы на рёбрах соответствуют действиям с моделью. Существуют ставшие классическими подходы к поиску в игровых деревьях – например, поиск в дереве методом Монте-Карло (Monte-Carlo Tree Search, MCTS). Кроме того, MCTS так же удавалось значительно усовершенствовать с помощью МО [32–34].

В качестве комплексного решения решаемых проблем со стратегией обхода дерева синтеза и сбором тренировочных данных рассматривался вопрос о возможности использования некоторых известных решений из предметной области игрового ИИ «как есть». К сожалению, обнаружено, что это невозможно по рассмотренным далее причинам.

Как успешное решение в общем виде задачи об ИИ для игр с ходами в 2018 году широкий резонанс в научно-исследовательском сообществе получила программа AlphaZero, созданная компанией DeepMind. Согласно данным Google Scholar на май 2023, препринт исследования [35] был процитирован в 1729 других публикациях, а соответствующая статья в журнале Science [33] – в 3235.

В основе подхода AlphaZero – упомянутый ранее MCTS, краткая иллюстрация сути которого приведена на рисунке 5. Процесс поиска в AlphaZero тоже двигался от корня к листьям, состоял из нескольких этапов, включал в себя случайную симуляцию игры там, где данные о дереве игры кончались, и обратно распространял полученный результат симуляции по дереву вверх, накапливая статистические знания. Затем этапы повторялись сначала, с корня, то есть процесс был стохастическим и итеративным, из-за чего в названии и присутствует отсылка к методу Монте-Карло. Чем дольше работал алгоритм поиска, тем больше статистических данных о дереве игры накоплено, тем точнее можно было делать предсказания о выгодности ходов при следующих играх.

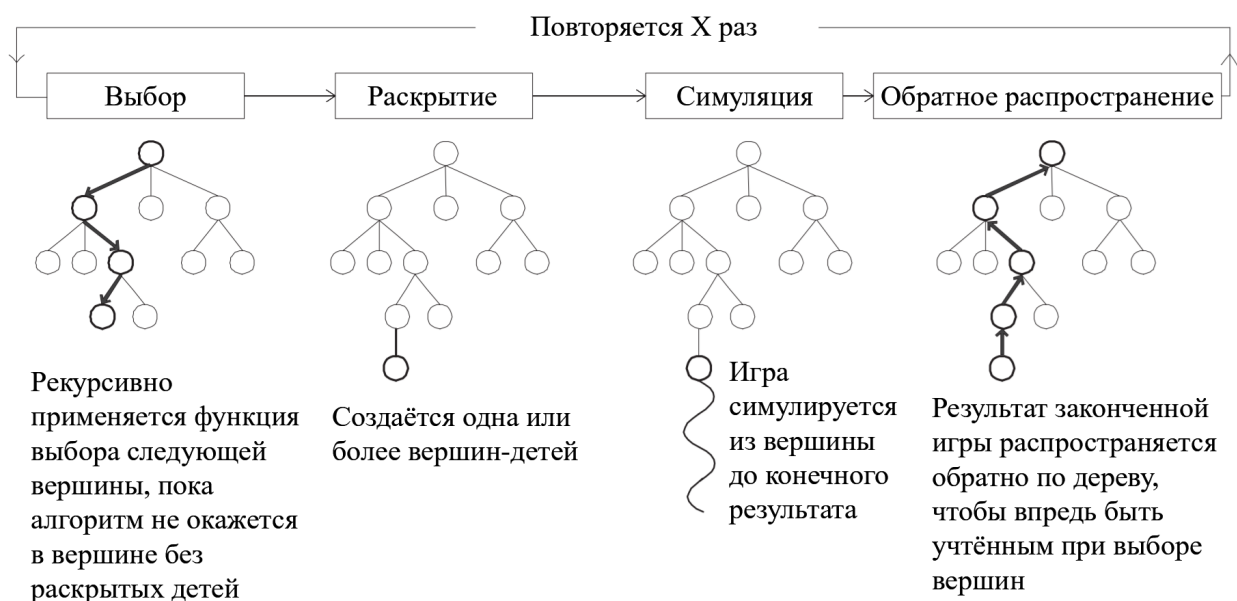


Рисунок 5 – Краткая демонстрация метода MCTS, воспроизведена из ВКР бакалавра автора [13]

Более подробно метод описан в [13], он в значительной степени уже идеологически повлиял на существующую реализацию прототипа метода синтеза с МО для САПР, но в рамках настоящей работы внимание прежде всего уделено надстройкам над MCTS, появившимся именно в AlphaZero и ранее подробно не рассмотренным.

Для ускорения MCTS авторы из DeepMind использовали непрерывно тренируемую и сразу используемую (в стиле обучения с подкреплением) глубокую нейронную сеть с единым «телом» и двумя «головами»: «policy head» и «value head». Разные «голова» обучались давать ответы о различных аспектах игры.

Policy-подсеть отвечала за выбор наилучшего следующего хода из возможных. При выборе хода дилемма разведки и эксплуатации (проблема балансирования между «жадным» применением известных знаний и открытием новых ценной возможных ошибок)

решалась с помощью методики «Upper Confidence Bounds Applied to Trees» – UCT [36]. Когда текущий участок дерева и вершин-кандидатов плохо исследован, процесс поиска при оценке выгоды ходов предпочитал больше «верить» policy-подсети. Со временем счётчик посещений конкретных вершин-кандидатов возрастал – собиралось больше достоверных статистических данных о данном участке дерева игры и о том, к победе или поражению вероятнее всего приведёт конкретный ход. UCT-оценка вершин-кандидата начинала в большей степени математически склоняться в сторону достоверно известной вероятности достижения победы из вершины-кандидата (отношение побед к общему количеству посещений вершины).

В свою очередь, value-подсеть отвечала за оценку «выгоды» того или иного состояния игры. Так авторы AlphaZero предложили сэкономить вычислительные мощности исключением дорогостоящих полноценных случайных симуляций игры из метода MCTS (см. этап «Симуляция» на иллюстрации к MCTS выше). Вместо симуляций использовались оценки нейросети, которые в процессе обучения сходились к достаточно точным.

AlphaZero продемонстрировала достойные результаты и переносимость между различными играми, достигая за часы или дни тренировки сверхчеловеческого уровня не только при игре в го, но и в шахматы, и в сёги. Подход подавался как обобщаемый на любые подобные игры, в связи с чем рассматривалась возможность применить AlphaZero «как есть» в качестве нового способа организации синтеза в САПР на основе МО, адаптировав постановку задачи к игровой. Такое применение повлекло бы за собой и обсуждаемую переработку стратегии обхода дерева синтеза, и переработку сбора тренировочных данных.

К сожалению, выявлены расхождения в деталях постановки задачи, которые делали адаптацию решения крайне затруднительным. Первое и самое существенное препятствие – неочевидность реализации отката процесса синтеза вверх по дереву, что должно стать ключевой способностью новой стратегии обхода. В играх подобные отмены ходов, как правило, не разрешены, поэтому AlphaZero выглядит неприспособленной к решению подобной задачи.

Алгоритмическое возвращение в корень при начале новой итерации MCTS во время обучения AlphaZero при этом откатом в интересующем смысле не является. В контексте решаемой задачи в области синтеза в САПР спецвычислителей откатом считается перемещение по дереву синтеза в сторону корня, аналогичное переходу по ребру в связи с принятием решения о совершении действия с моделью вычислителя. Решение о совершении отката должно быть принято в ходе рассмотрения возможных вариантов продолжения процесса синтеза. В случае MCTS процесс возвращается в корень как в исходное положение для продолжения обучения.

Теоретически можно попытаться рассматривать откат назад как ещё один возможный ход из каждого состояния, но, чтобы дерево игры (или синтеза) оставалось корректным (не содержало циклов), необходимо воспринимать вершины, в которые ведёт данный ход отката, как абсолютно новые вершины, копирующие оригинальные. Это сопряжено со значительными техническими сложностями, делает любое дерево синтеза гарантированно бесконечным и распределяет ценную статистическую информацию о перспективности вершин, то есть ценности исследования их поддеревьев, по бесконечному количеству копий этого поддерева. Например, статистика «побед» из какой-либо вершины A будет никак не связана со статистикой вершины A' , в которую можно попасть спуском в какого-либо ребёнка вершины A и откатом на один ход обратно. Такие потери информации не соответствуют «физическому смыслу» деревьев игр или синтеза и крайне нежелательны.

Также требуется запоминать уже посещённые вершины и исключать их из копий дерева, что технически накладно. Подобное исключение обязательно, так как модель будет склонна повторно требовать спуск в вершину, из которой только что откатилась, ведь она уже была оценена как наиболее привлекательный кандидат ранее. Как следствие, процесс будет бесконечно циклично «блуждать» в данной области дерева, если не применять описанное исключение посещённых вершин. Таким образом, откаты вверх по дереву и AlphaZero совместить вместе оказалось крайне проблематично.

Второе препятствие при применении AlphaZero «как есть» состоит в том, что игровой ИИ тренируется и используется на одном и том же постоянном дереве игры. Это заложено в корне самого алгоритма MCTS: результирующая стратегия после обучения управляется преимущественно счётчиками побед и посещений на конкретных вершинах дерева игры. В случае САПР спецвычислителей для каждого целевого алгоритма дерева синтеза новое, а оценочная модель МО нужна универсальная, то есть применимая к деревьям, которые раньше никогда не видела. Предполагается, что между этими деревьями будут инвариантны закономерности, как правило, приводящие к успешным или неуспешным результатам синтеза, которые от модели и требуется выделить при обучении.

Третье препятствие – непрерывность и параллельность процессов симуляции игры, сбора тренировочных данных и тренировки модели. Всё это производится одновременно, AlphaZero в процессе тренировки «играет сама с собой». Однако в применении к САПР спецвычислителей такой подход вычислительно слишком дорог: поддерживать работающие копии САПР NITTA с различными деревьями синтеза и на ходу генерировать тренировочные экземпляры, как минимум, на порядок замедлит обучение модели МО относительно текущего подхода, демонстрирующего тренировочные экземпляры модели тысячами в секунду.

Есть и другие, менее существенные различия в постановке задачи: например, в процессе синтеза не происходит игры соперников против друг друга. Подробное рассмотрение таких несоответствий опущено, так как они не влияют на фундаментальную применимость подхода, в отличие от рассмотренных ранее проблем.

Таким образом, применить AlphaZero «как есть» не представляется возможным. Требуется спроектировать оригинальное решение поставленных проблем.

2.2 Проектирование и реализация новой стратегии обхода

К реализации откатов в процессе синтеза выявлено два принципиально разных подхода. Первый – рассматривать откат по дереву синтеза обыкновенным ребром. Этот вариант уже частично рассмотрен выше, так как он получается при попытке применить AlphaZero, добавив в игровое дерево поддержку отмены ходов.

Показано, что такой подход технически сложен в реализации. Кроме того, он исключительно по техническим причинам ограничивает мобильность процесса синтеза при перемещении по дереву. Допустим, стоит задача научить модель МО пошагово переходить в следующую по перспективности вершину. Предположим, что эта вершина доступна из текущей позиции на дереве по такому пути: «13 раз перейти в родителя, затем спуск в ребёнка №15, затем в №32, затем в №4». Чтобы корректно пройти этот путь, модели необходимо выбрать верное действие 16 раз, а на каждом шаге вариантов может быть больше сотни. Добиться корректного поведения будет проблематично.

Вследствие описанных проблем было решено отказаться от первого подхода в пользу второго. Если натренировать модель МО на оценку перспективности вершины (схоже с value-подсетью в AlphaZero) так, чтобы различные оценки перспективности вершин в дереве были сравнимы между собой, то можно обходить вершины в порядке убывания их оценки. При таком подходе модель МО влияет на производимые откаты и может обучаться их провоцировать (основная сложность заключается в инженерии тренировочного набора данных), но модели не приходится прокладывать путь по дереву синтеза.

Техническая реализация отличается простотой. Можно использовать существующие в САПР NITTA механизмы по оценке вершин оценочной функцией. Впрочем, для синтеза на основе МО требуется адаптировать тренировочный набор данных (больше деталей на этот счёт в разделе 3).

Сама стратегия может быть описана следующим образом. В процессе синтеза поддерживается единый на всё дерево упорядоченный список вершин-кандидатов к посещению по убыванию оценки (рисунок 6). Вершины посещаются по этому списку сверху вниз. При посещении вычисляются их вершины-дети, они оцениваются и добавляются в список, не нарушая упорядоченности списка по убыванию оценок.

На рисунке идентификатор посещённых вершин в таблице отмечен ярко-красным цветом, а совершённый путь на дереве показан ярко-красными стрелками. Зелёные вершины соответствуют вершинам с высокими оценками, тёмно-красные вершины – вершинам с низкими оценками. По состоянию дерева видно, что процесс синтеза оказался в бесперспективной ветви и что все вершины-дети имеют низкую оценку (красные).

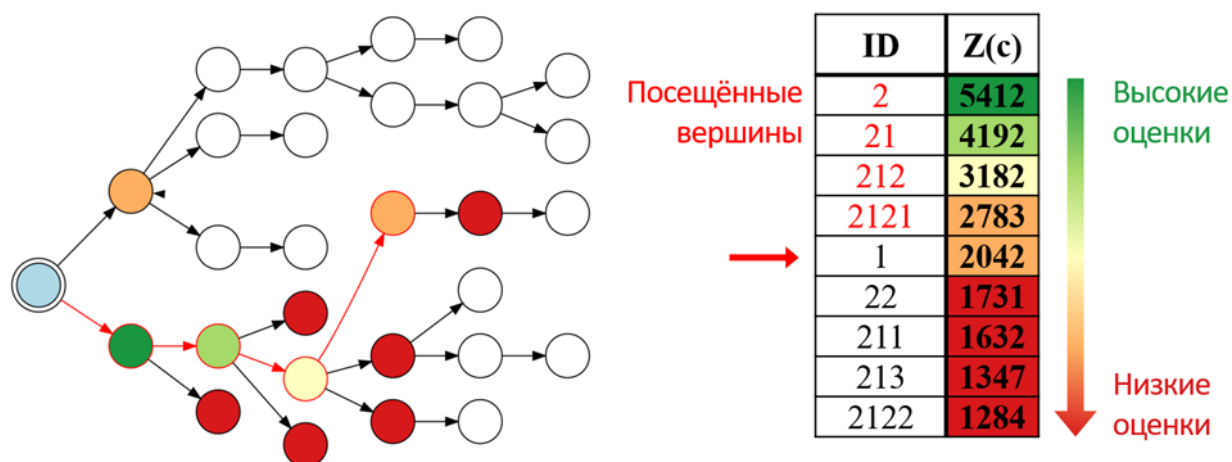


Рисунок 6 – Новая стратегия обхода дерева синтеза, упорядоченный по убыванию оценок список вершин-кандидатов

На следующую к посещению вершину в таблице указывает большая красная стрелка рядом с таблицей. Вершина имеет идентификатор 1, это верхний ребёнок корня. Так как он является следующим в упорядоченном списке, то есть его оценка выше всех непосещённых вершин-кандидатов из нижней бесперспективной ветки, далее будет посещен именно он. Фактически это является откатом, причём сразу в другую ветку. Будут вычислены и добавлены в список дети вершины с идентификатором 1, у них будут потенциально более высокие оценки, и, если так, то спуск, в соответствии с упорядоченным списком, продолжится именно в них (рисунок 7).

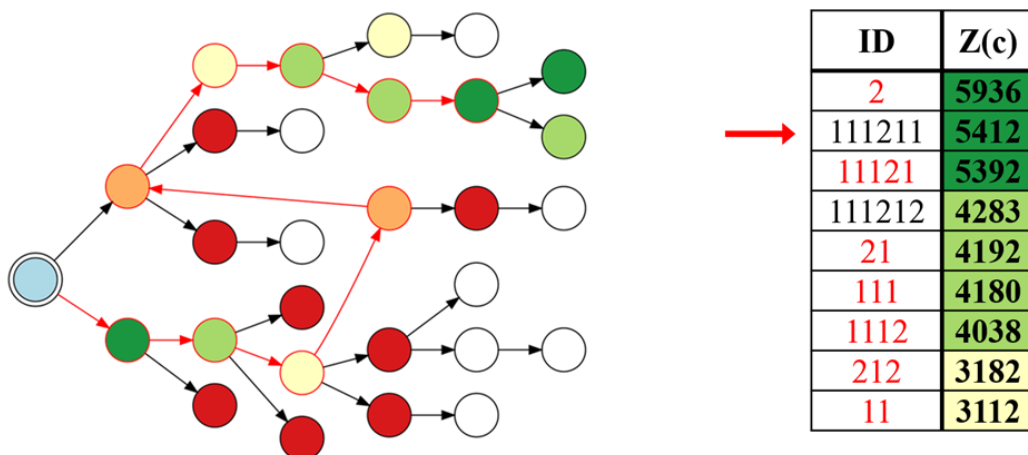


Рисунок 7 – Новая стратегия обхода откатил процесс синтеза, спуск продолжается в ветку с более высокими оценками

Таким образом, бесперспективная ветка была заброшена, стратегия обхода выполнила свою задачу. Технически эффективно поддерживать упорядоченным подобный крупный список с постоянно используемыми операциями вставки и извлечения элементов возможно с помощью очереди с приоритетами, реализованной с помощью структуры данных «куча».

Дополнительным преимуществом предлагаемого подхода является тот факт, что он без дополнительных усилий применим не только к методу синтеза на основе МО, где значения оценочной функции определяются моделью, но и к любым другим оценочным функциям, значения которых сравнимы между собой для вершин в различных частях дерева, в том числе не на основе МО.

Важная деталь, которую стоило предусмотреть – необходимость дополнительно мотивировать процесс синтеза спускаться к листьям дерева, сделав оценку детей посещаемой вершины функцией её глубины. Вводится понятие *приоритета вершины* p_A :

$$p_A = s_A * b^{d_A}, \quad (1)$$

где d_A – глубина вершины A, s_A – оценка вершины A, а b – параметр стратегии обхода, регулирующий стремление предпочитать поиск в глубину поиску в ширину (обычно около 1,2). Чем больше b , тем быстрее с ростом глубины вершины d_A будет (экспоненциально) расти p_A .

Чтобы приоритет влиял на стратегию обхода, необходимо теперь упорядочивать вершины в убывающем списке не по их оценкам, а по приоритету. Тогда чем больше b , тем быстрее стратегия обхода будет спускаться к листьям дерева, но, с другой стороны, тем

больше должна будет упасть оценка новых вершин, встречаемых на пути, для провоцирования отката.

Усовершенствованная стратегия обхода реализована на языке Haskell как часть кодовой базы САПР NITTA и получила название TopDownScoreSynthesisIO. Исходный код доступен в публичном репозитории проекта [30]. В качестве упорядоченного списка использовалась структура данных «куча». Реализовано соответствующее журналирование, а также возможность указать максимальное количество шагов.

Добавлены соответствующие аргументы интерфейса командной строки (command-line interface, CLI), позволяющие выбирать, во-первых, стратегию обхода дерева синтеза, а во-вторых – задавать для новой стратегии обхода описанный выше параметр b .

2.3 Испытание разработанной стратегии обхода

Для проведения эксперимента были реализованы необходимые модули на языке Python 3. САПР NITTA запускалась в качестве подпроцесса с различными CLI-аргументами, меняющими стратегию обхода, параметр b и синтезируемый целевой алгоритм. Среди измерявшихся величин:

- общая длительность процесса синтеза в секундах,
- количество посещённых вершин,
- длительность синтезированной управляющей программы в тактах,
- глубина листа-результата синтеза в дереве.

Для испытаний использовано 11 существующих примеров целевых алгоритмов, а также написан ещё 21 новый. Итого использовалось 33 целевых алгоритма, 5 из них имели небольшие деревья синтеза (до 6000 вершин), остальные имели деревья с сотнями тысяч, миллионами и десятками миллионов вершин.

Целевой алгоритм считался успешно синтезированным, если его процесс синтеза успел завершиться за 60 секунд и был получен успешный результат. Давалось 3 попытки, собранные величины агрегировались среди успешных попыток (среднее, среднеквадратичное отклонение, минимум, максимум). Сравнивались стратегии обхода «StateOfTheArtSynthesisIO» (сокращённо – SOTA) и новая «TopDownScoreSynthesisIO» с параметрами $b = 1,4$ и $b = 1,2$ (сокращённо – TD-1.4 и TD-1.2, соответственно).

При применении новой стратегии обхода общее количество успешных запусков синтеза могло незначительно возрасти (на 3 для стандартной оценочной функции с TD-1.4),

могло незначительно упасть (на 5 для стандартной оценочной функции с TD-1.2), могло остаться таким же (актуальная версия метода синтеза с МО, стратегии TD-1.2 и TD-1.4).

Количество посещённых вершин в процессе синтеза значительно упало (среднее по максимумам из 3 запусков упало с 2935 до 1759, стандартная оценочная функция, $b = 1,4$). Среднее время успешного синтеза по всем целевым алгоритмам упало с 5,7 с до 2,0 с (стандартная оценочная функция, $b = 1,4$).

На некоторых примерах целевых алгоритмов с большими деревьями синтеза (десятки миллионов вершин), в частности – `sin_ident.lua`, среднее время синтеза с 14,02 с при применении новой стратегии обхода упало до 1,98 с и 1,56 с при $b = 1,2$ и $b = 1,4$ соответственно – более чем в 6 раз (стандартная оценочная функция). Для алгоритмов `physics2.lua` (сопоставимый с `sin_ident.lua` размер дерева синтеза) и `physics3.lua` (дерево синтеза больше предыдущих примеров, глубина листов достигает 160) результаты аналогичны. Больше примеров представлено на таблице 1.

Таблица 1 – Сравнение среднего времени синтеза некоторых целевых алгоритмов при использовании различных стратегий обхода

Целевой алгоритм	Среднее время синтеза, с		
	SOTA	TD-1.2	TD-1.4
<code>sin_ident.lua</code>	14,0	2,0	1,6
<code>physics2.lua</code>	10,8	0,8	0,7
<code>physics3.lua</code>	43,7	45,2	9,1
<code>cyclic20-tight.lua</code>	42,6	–	2,5
<code>cyclic5.lua</code>	1,0	0,6	0,5
<code>vars.lua</code>	1,0	0,6	0,5
<code>pid.lua</code>	0,9	0,6	0,5
<code>matrix-mult-1x3.lua</code>	0,5	0,4	0,4

Что касается длительности синтезированных управляющих программ в тактах, то, в целом, замена стратегии обхода на неё не повлияла. Например, при применении стандартной оценочной функции средняя длина составила 23.3 такта для стратегии обхода SOTA, 21.9 для TD-1.2 и 25.8 для TD-1.4. Если в ходе обхода дерева обнаруживалось несколько успешных листов (стратегия обхода SOTA предполагала такую возможность), то выбирался наилучший в соответствии с критерием оценки эффективности лист, то есть с минимальной длительностью.

2.4 Выводы

Новая стратегия обхода дерева синтеза позволила добиться существенного ускорения процесса синтеза для многих протестированных целевых алгоритмов, в особенности – для сложных алгоритмов с деревьями синтеза большого размера (миллионы вершин). Негативного влияния на качество результатов синтеза (длительность управляющей программы) обнаружено не было. Таким образом, новая стратегия обхода дерева синтеза успешно повысила эффективность САПР в соответствии с обозначенными критериями.

Параметр b для достижения наилучших результатов может требовать ручной настройки под конкретный целевой алгоритм. Как правило, $b \in [1,1; 1,5]$. Для стандартной оценочной функции обычно лучших результатов позволяет достичь $b = 1,4$, нежели $b = 1,2$, для метода синтеза на основе МО – наоборот.

В то же время обнаружено, что новая стратегия обхода вследствие экспоненциального роста p_A в больших деревьях синтеза не склонна делать откаты далеко назад по дереву, которые иногда могут быть необходимы для успешного синтеза. В качестве направления дальнейших исследований можно выделить изучение возможности реализации «далеких» откатов с помощью специально введённого алгоритмического механизма, переопределяющего приоритет вершин в определённых условиях. Например, если обход «застрял» на определённой глубине и натывается на много листов с неуспешным результатом синтеза. Есть основания полагать, что такая доработка может положительно повлиять на способность синтезировать сложные целевые алгоритмы, которые сейчас не поддаются синтезу ни одной из стратегий обхода.

Исходный код реализации новой стратегии обхода дерева синтеза в составе САПР NITTA и модули для проведения описываемых испытаний опубликованы в открытый доступ в репозитории САПР NITTA [30], где находится и полная сводная таблица с результатами испытания [37].

3 СБОР ТРЕНИРОВОЧНЫХ ДАННЫХ ИТЕРАТИВНЫМ МЕТОДОМ

Данный раздел посвящён усовершенствованию метода сбора тренировочного набора данных для обучения модели МО, которая будет использоваться в качестве оценочной функции в процессе синтеза. В первом подразделе осуществляется обзор существующего метода сбора данных, реализованного автором ранее в [13]. Затем этот метод анализируется с целью выявления его достоинств, которые желательно сохранить при проектировании нового метода, а также недостатков, которые новый метод будет должен исправить. Сформулированы конкретные требования к усовершенствованному методу.

Далее на основе полученной в ходе анализа информации новый метод спроектирован. Рассмотрено, как он решает поставленные задачи. В последних подразделах описывается реализация спроектированного усовершенствованного метода и её испытание в соответствии с определёнными ранее критериями. Обозначены полученные результаты и возможности дальнейшего совершенствования разработанного метода.

3.1 Ревизия существующего метода сбора тренировочных данных

Чтобы оценочная функция способствовала повышению эффективности САПР, тренировочные метки должны быть пропорциональны ценности посещения оцениваемой вершины и, соответственно, исследования её поддерева. Достигнуть этого ранее удалось с помощью вычисления оценки листьев на основе *метрик листьев* и обратного распространения оценки вверх по дереву [13], что также напоминает некоторые идеи из MCTS. Метрики листьев численно характеризовали, насколько полученный в листе результат синтеза соответствовал желаемому (в качестве примера желаемыми считались корректные управляющие программы с наименьшей длительностью в тактах). Если в листе получен неуспешный результат синтеза, то метрики листьев не использовались вовсе, так как меткой устанавливалась численная константа, соответствующая очень низкой оценке вершины. С помощью обратного распространения оценок листьев могла быть рассчитана оценка для каждой вершины в дереве на основе оценок в её поддерева.

Есть связанная с описанным подходом сложность: деревья синтеза для различных целевых алгоритмов могут сильно различаться между собой и по средней глубине листьев, и по длительностям получаемых управляющих программ, и по другим возможным

метрикам листьев. Однако важно, чтобы метки листьев, вычисляемые на основе метрик, не зависели от синтезируемого целевого алгоритма, так как тренируемая модель должна быть универсальной.

Алгоритмически проблема решена отказом от однократного обхода дерева синтеза в пользу многократного, что сделало возможным технически несложно реализовать *нормализацию* метрик листьев – корректирующее преобразование распределений метрик листьев различных деревьев, приводящее распределения к единообразию с помощью центрирования значений относительно математического ожидания выборки и компенсации среднеквадратичного отклонения. Математические детали приведены в [13].

На рисунке 8 упрощённо проиллюстрирована суть операции. Предположим, собраны данные с трёх различных деревьев и получены значительно отличающиеся распределения какой-либо одной метрики листьев. Нормализация позволяет преобразовать полученные абсолютные значения к их относительным представлениям (справа). Алгебраическое сравнение представлений метрики между различными деревьями будет гораздо более осмысленным с точки зрения предметной области, что математически упрощает тренировку модели, применимую на многих деревьях сразу.

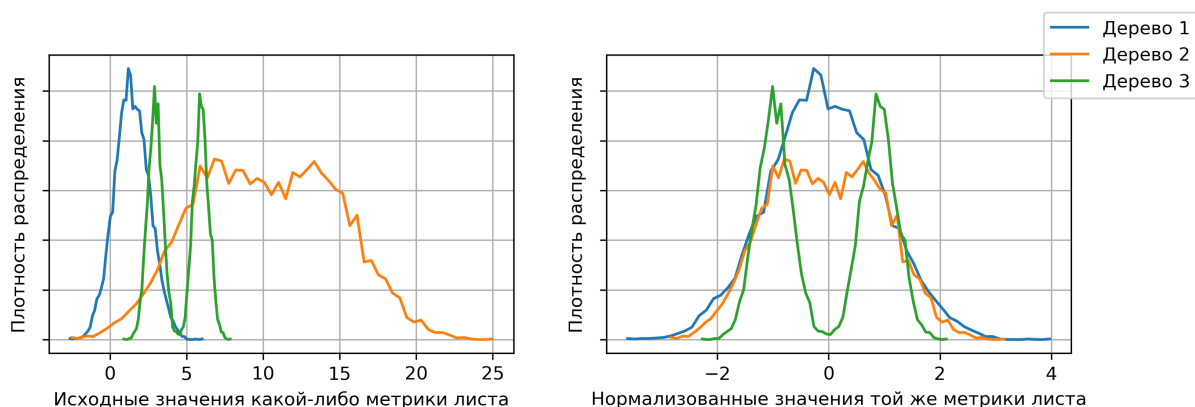


Рисунок 8 – Нормализация значений одной метрики листьев, вычисленной для разных деревьев синтеза с различными её распределениями

Таким образом, существующий многопроходный процесс сбора данных состоит из нескольких последовательных этапов. Результаты работы предыдущего этапа необходимы для начала следующего. Ключевые этапы процесса сбора данных в применении к одному целевому алгоритму могут быть кратко описаны следующим образом:

- 1) вычисление полного дерева синтеза данного целевого алгоритма;
- 2) расчёт нормализованных метрик для всех листьев дерева;
- 3) генерация для каждой вершины дерева тренировочного экземпляра вида «характеристики вершины – метка» с вычислением меток, начиная с листьев;

4) сохранение сгенерированного на диск.

Среди достоинств существующего метода – относительная простота реализации, детерминированность процесса, гарантированно равномерное и полное покрытие дерева синтеза при переработке в тренировочные экземпляры. Такие преимущества, кроме того, обеспечивают возможность достоверно определить размеры перерабатываемых деревьев, а многопроходность алгоритма позволяет иметь точные оценки статистических параметров (моментов распределений) метрик листьев при вычислении абсолютно всех меток, включая обрабатываемые первыми.

Однако есть один очень существенный недостаток: чтобы начать перерабатывать дерево синтеза в экземпляры, необходимо прежде полностью его вычислить. По отмеченным ранее причинам (прежде всего, неспособность некоторых деревьев помещаться в оперативную память) не всегда это возможно, особенно если речь заходит о потенциально бесконечных деревьях.

Кроме того, исследования автора в рамках производственных практик показали, что сбор данных с одного «большого» дерева проблематично полностью параллелизовать, возможна только частичная параллелизация в рамках одного этапа процесса на одной машине, что ограничивает применение вычислительных кластеров.

Также выявлена ещё одна потенциально неудобная особенность: процесс нельзя произвольно запускать и останавливать (например, когда собрано достаточно данных с дерева синтеза). Для переработки дерева процесс должен быть успешно выполнен с начала и до конца, после чего на диск будут записаны либо экземпляры со всего дерева сразу, либо ничего, хоть это и скорее особенность текущей реализации. Из этого также следует, что при возникновении ошибок придётся начинать процесс сначала (впрочем, отлаженный код работает на данный момент исправно, так как для основных ошибок написаны специальные обработчики).

Таким образом, обнаружена необходимость в проектировании нового метода сбора тренировочных данных, от которого требуется:

- возможность переработки «больших» и бесконечных деревьев;
- поддержка полной параллелизации обработки одного дерева вплоть до возможности осуществления параллельной полезной работы на разных машинах;
- возможность произвольной остановки процесса в любой момент с сохранением частичных результатов на диск.

3.2 Проектирование итеративного метода сбора данных

Обзор AlphaZero в предыдущем разделе настоящей работы представлен в том числе потому, что удалось заимствовать некоторые подходы и для реализации нового метода сбора тренировочных данных, ведь исследователи из DeepMind решили, в целом, схожую задачу и обучают нейронную сеть на данных, собираемых с деревьев игр очень большой мощности. Им это удалось сделать благодаря стохастической итеративной природе процесса обучения, унаследованной от MCTS.

3.2.1 Итеративность и стохастичность нового метода сбора данных

Под итеративностью понимается разбиение процесса на небольшие самодостаточные итерации. Каждая итерация включает в себя исследование относительно небольшой части дерева, сбор тренировочных данных с этой части и их обработку. Этот подход кардинально отличается от нескольких последовательно выполняемых глобальных этапов, на которые разделён существующий процесс сбора данных. Итеративность является подходящим для САПР спецвычислителей решением к проблеме сбора данных с больших деревьев, так как каждая итерация требует наличия в оперативной памяти только своего участка дерева и производит готовые тренировочные экземпляры. Благодаря итеративности же можно достигнуть и возможности останавливать или возобновлять процесс сбора данных в любой момент, так как существует промежуточный прогресс, который можно записать на диск.

Стохастичность позволяет упростить реализацию итеративного метода. Предлагается в каждой итерации производить случайный спуск от корня к какому-либо листу, вычислять метку листа и осуществлять обратное распространение метки по вершинам в пути до корня. Случайность спуска побуждает неполное покрытие дерева стремиться к равномерному и подразумевает отсутствие необходимости координировать итерации между собой (планировать и распределять работу), что в особенности значительно упрощает параллелизацию процесса между несколькими машинами.

Таким образом, подобный итеративный процесс будет соответствовать всем поставленным ранее требованиям. Однако есть ряд нерешённых проектировочных проблем, требующих рассмотрения.

3.2.2 Проблема итеративной нормализации метрик листьев

Одна из проблем исходит из того, что для вычисления нормализованных метрик листа нужны выборочные математическое ожидание и среднеквадратичное отклонение для каждой метрики. В старом методе они вычислялись по известному полностью вычисленному дереву, но теперь такой возможности нет.

Предлагается накапливать значения, собранные за предыдущие итерации, и вычислять указанные статистические характеристики распределений значений метрик листьев на основании данных, собранных к данному моменту. Существуют соответствующие формулы (2), позволяющие это делать:

$$S_1 = \sum_{i=1}^n x_i, \quad S_2 = \sum_{i=1}^n x_i^2, \quad \mu = \frac{S_1}{n}, \quad \sigma = \sqrt{\frac{S_2}{n} - \left(\frac{S_1}{n}\right)^2}, \quad (2)$$

где n – количество собранных значений метрики, x_i – собранное значение метрики, μ – выборочное математическое ожидание собранных значений метрик, σ – выборочное среднеквадратичное отклонение собранных значений метрик, S_1, S_2 – обновляемые при сборе суммы, из которых можно по запросу вычислить μ и σ . В процессе сбора данных мощность выборки собранных значений метрик n будет расти, вместе с чем μ и σ будут становиться ближе к истинным.

3.2.3 Проблема коллизии путей и агрегирование данных

Есть ещё одна проектировочная проблема. Постоянные повторы итераций, начинающих спуск из корня, и случайность выбираемого пути порождает возможность повторного посещения одних и тех же вершин в разных итерациях. Далее это явление называется *коллизией путей*. Так как такие вершины стали частью нескольких путей, то к ним будут обратно распространены метки от нескольких листьев.

Необходимо спроектировать, каким образом вычислять финальную метку для этой вершины. Схожая проблема уже решалась ранее в [13], но для полных поддеревьев: использовалась взвешенная сумма максимальной и средней меток листа из поддерева. В проектируемом случае отличие состоит лишь в том, что поддерево неполное и известно не сразу.

К сожалению, это вынуждает производить дедупликацию вершин в собранных необработанных тренировочных экземплярах и агрегацию их меток по формуле из [13] уже

после окончания итеративного процесса. В то время как такое агрегирование можно считать возвратом к многоэтапному процессу сбора данных, оно не ограничено оперативной памятью, может быть реализовано асимптотически быстрым алгоритмом и работать с уже записанными на диск данными, поэтому не наследует недостатки предыдущего метода. Стоит отметить, впрочем, что коллизия путей не должна быть частой вдалеке от корня больших деревьев, потому что вероятность повторить путь с увеличением его длины падает экспоненциально.

3.2.4 Предполагаемые недостатки итеративного метода сбора данных

Из предполагаемых недостатков спроектированного подхода можно отметить, что процесс недетерминированный и более сложный в реализации, а также вследствие случайности процесса проблематичнее оценить долю посещённых вершин в нём и определить, когда собрано достаточно данных и можно останавливать сбор.

Однако в свете достоинств нового метода, его полного соответствия поставленным требованиям и новым возможностям, которые появляются с его использованием, принятие указанных недостатков считается оправданным. Дальнейшая работа посвящена реализации и испытанию спроектированного итеративного метода сбора данных.

3.3 Реализация итеративного метода сбора данных

Кодовая база САПР NITTA и разработанного ранее прототипа метода синтеза с МО была модифицирована, чтобы реализовать спроектированный метод сбора тренировочных данных. Использовался язык Python 3, так как на нём был реализован совершенствуемый прототип синтеза на основе МО. Для выявления дефектов и отладки применялось автоматическое модульное тестирование с использованием пакета `pytest`.

Реализован CLI к новому функционалу. На рисунке 9 представлен пример вызова «помощи» (встроенного руководства по эксплуатации). Вся разработанная документация написана на английском языке в соответствии с правилами международного проекта, каким является САПР NITTA.

```
devuser@42631910de77:/app$ python ml/synthesis/src/scripts/crawl_data_by_tree_sampling.py -h
usage: crawl_data_by_tree_sampling.py [-h] [-n N_SAMPLES] [-w N_WORKERS] [-i N_NITTAS]
                                     [--nitta-path NITTA_PATH]
                                     file
```

Gathers training data for provided input file by sampling its synthesis tree.

positional arguments:

file Path to the input file for NITTA to synthesize.

optional arguments:

-h, --help show this help message and exit

-n N_SAMPLES, --n-samples N_SAMPLES

Number of samples to gather. A sample is a full descent from root to leaf of the synthesis tree. Default: 5000 (adjust to tree size!).

-w N_WORKERS, --n-workers N_WORKERS

Number of workers to use for parallel sampling. 1 recommended in most cases. Default: 1.

-i N_NITTAS, --n-nittas N_NITTAS

Number of NITTA instances to run. 1 recommended in most cases. Default: 1.

--nitta-path NITTA_PATH

Path to the NITTA executable. Default: 'stack exec nitta' is used.

It's recommended to adjust the number of samples to the size of the tree, as well as other parameters to your machine's capabilities.

Рисунок 9 – Пример вызова руководства по эксплуатации с помощью разработанного CLI для нового функционала

Реализован в общем виде и встроен в процесс сбора данных итеративный сборщик метрик листьев в соответствии с формулой (2). Учтены и другие требовавшие внимания аспекты разрабатываемого решения, рассмотренные далее.

3.3.1 Затухание тренировочных меток при обратном распространении

Чтобы дополнительно стимулировать новую стратегию обхода спускаться к листьям, если модель МО считает, что они близко, в рамках усилий по инженерии тренировочных данных реализовано затухание меток при их обратном распространении. Самый простой способ добиться такого эффекта – умножить метку на коэффициент затухания при каждом шаге обратного распространения (переход от вершины-ребёнка к вершине-родителю). Затухание получится экспоненциальным, что соответствует желаемому.

Также был учтён тот факт, что деревья могут быть разной глубины, поэтому коэффициент затухания важно не задавать в качестве константы, а вычислять на основе текущей оценки средней глубины листьев в дереве. В связи с этим настройка агрессивности затухания реализована указанием желаемой доли метки листа, которая должна попасть в метку корневого узла.

3.3.2 Параллелизация итеративного процесса сбора данных

Также потребовала дополнительных усилий разработка параллелизованной версии нового процесса сбора данных, так как способность разработанного средства к параллелизации ограничивается особенностью используемой реализации языка Python (CPython), а именно – глобальной блокировкой интерпретатора (Global Interpreter Lock, GIL), что предотвращает использование нескольких потоков для ускорения вычислительно дорогостоящих операций с нагрузкой преимущественно на центральный процессор. С учётом этого было реализовано несколько других механизмов, позволяющих сделать процесс сбора данных параллельным.

Во-первых, реализована мультипроцессная параллелизация с помощью библиотеки `joblib`, управляющей несколькими копиями интерпретаторов Python и межпроцессным взаимодействием с ними. В ответственность библиотеки входит организация очереди программных задач, вызов необходимых вычислений, а также сериализация и десериализация пересылаемых данных между процессами.

Во-вторых, основной код, отвечающий за получение данных из САПР NITTA с помощью сетевого взаимодействия, написан в стиле, поддерживающем невытесняющую многозадачность (использованы ключевые слова `async` и `await`, функции модулей `asyncio` и `aiohttp`). Данный вид параллелизации в CPython эффективен в отношении задач, в которых преимущественно осуществляется ожидание на операциях ввода-вывода. Случай сбора данных под это описание подходит, так как ожидание ответа от САПР, производящей значительную часть вычислений, происходит именно при сетевом вызове.

В-третьих, реализована возможность использовать несколько копий процессов САПР NITTA для предотвращения простоя центрального процессора из-за потенциальных блокировок на критических секциях внутри самой САПР. Выяснить, какой из этих методов эффективнее всего ускорял сбор данных, предстояло при испытаниях.

3.3.3 Аппроксимация покрытия дерева синтеза при сборе данных

Также была осуществлена попытка смягчить некоторые недостатки метода, выявленные при проектировании, а именно реализовать аппроксимационный оценщик *покрытия дерева* γ после исполнения процесса сбора данных новым методом. Покрытие дерева определяется следующим образом:

$$\gamma = \frac{n_B}{N}, \quad (3)$$

где n_B – количество уникальных посещённых вершин в процессе сбора данных, а N – общее количество вершин в обрабатываемом дереве синтеза (заранее неизвестно при итеративном стохастическом сборе данных). С помощью этой величины можно понять, насколько полно отражено обрабатываемое дерево в собранных данных.

К сожалению, известно только то, как меняется с течением процесса сбора количество уникальных посещённых вершин n_B и общее количество посещений вершин в ходе сбора данных n_{Π} (может быть больше N), из чего можно вычислить *долю коллизий* r_c :

$$r_c = \frac{n_{\Pi}}{n_B} - 1. \quad (4)$$

При таком определении значения r_c , близкие к 0, будут говорить о том, что уже посещённые вершины встречались в ходе сбора крайне редко. Чем выше значение – тем чаще такое происходило.

Идея состояла в следующем: собрать эмпирические данные о зависимости N (неизвестно при сборе) от r_c (известно при сборе), затем полученные эмпирические данные аппроксимировать функцией φ с целью получить математическую зависимость $\gamma = \varphi(r_c)$, которая, как предполагается, позволит оценить полученное в процессе сбора покрытие любого дерева синтеза лишь по известному r_c . Предполагается, что такая зависимость имеет неслучайную природу и будет объяснима из соображений математической статистики и теории вероятности, если аналитически решить задачу случайного спуска по дереву. Следовательно, зависимость должна быть универсальной и переносимой между деревьями (для этого аппроксимируется именно относительное γ вместо абсолютного количества вершин N).

Зависимость на практике оказалась достаточно сложной. Для упрощения аппроксимации осуществлялись множественные попытки математически преобразовывать r_c : $\ln(r_c)$, $\frac{r_c}{n_B}$ и многие другие. Также производились попытки аппроксимировать не непосредственно $\gamma = \varphi(r_c)$, а другие функции, которые затем можно было свести к φ .

На рисунке 10 изображена финальная экспериментальная версия аппроксимации представленной в (5) зависимости с эмпирическими данными (точками), рассчитанными для целевого алгоритма-примера counter.lua с небольшим деревом синтеза (примерно 5 тысяч вершин) из репозитория САПР NITTA.

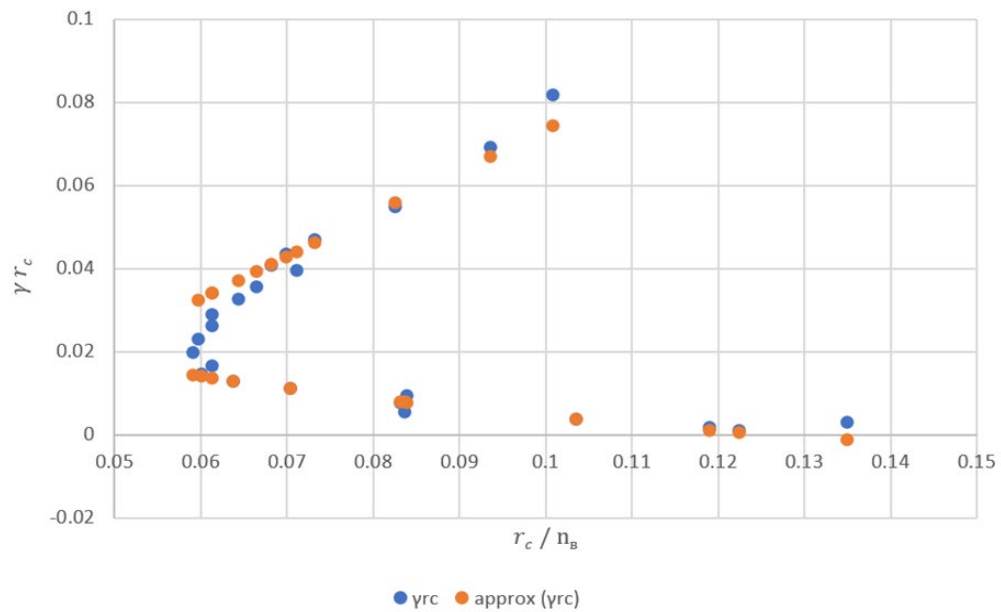


Рисунок 10 – График аппроксимации зависимости (5)

$$\gamma r_c = \chi \left(\frac{r_c}{n_B} \right). \quad (5)$$

Зависимость, как выяснилось, не является взаимно-однозначным соответствием – одному значению $\frac{r_c}{n_B}$ могут соответствовать несколько значений γr_c . Следовательно, аппроксимирующую χ пришлось задать кусочной функцией из двух частей: верхняя ветвь соответствует участку, где $r_c > 1$, и нижняя ветвь – участку с $r_c < 1$. Смещение ветвей в районе $r_c = 1$ производится с коэффициентом, управляемым сигмоидой.

Данная аппроксимация, хоть и, вероятно, является переусложнённой, лучше всех других испытанных переносилась между различными небольшими деревьями, где относительно быстро достигалась доля коллизий $r_c > 0,5$. Для более крупных деревьев такой показатель может оказаться недостижим и за несколько лет сбора данных.

При попытке использовать полученную зависимость для оценки покрытия новых крупных деревьев в случае $r_c \rightarrow 0$ выявлено, что нижняя ветвь была аппроксимирована неточно, поэтому получаемые оценки были неадекватны. Требуется сбор большего количества аппроксимированных данных для развития такого подхода. На данный момент дальнейших успехов в решении задачи оценки покрытия дерева достичь не удалось.

3.3.4 Улучшение качества кодовой базы

Кроме непосредственного обновления метода сбора данных, также были проведены работы по устранению технического долга. Кодовая база была переработана, разделены смешивающие различные ответственности классы (например, NittaNode). Устранены избыточные зависимости на функционально дублирующие друг друга библиотеки: например, использования `dataclasses-json` переработаны на уже применяемую в другой части проекта библиотеку `Rydantic`. Некоторые важные части разработанного кода были покрыты автоматизированными модульными тестами. Добавлены различные инструментальные средства для контроля качества кодовой базы проекта на языке Python:

- средство «black» для автоматического форматирования кода и контроля за соглашениями о стиле (соответственно, вся кодовая база была переформатирована);
- статический анализатор кода «mypy» для дополнительных проверок корректности и идиоматичности кода, в том числе с помощью используемых повсеместно в проекте подсказок о типах (`type hints`);
- средство «isort» для сортировки зависимостей модулей (`imports`);
- средство «autoflake» для удаления неиспользуемых зависимостей модулей.

Было проведено испытание разработанных решений. Их результаты отражены в следующем подразделе.

3.4 Испытание итеративного метода сбора данных

3.4.1 Ручное тестирование итеративного сбора данных

Проведено базовое ручное тестирование разработанного средства. На рисунке 11 приведён снимок экрана, демонстрирующий записанный на диск файл в формате CSV («значения, разделённые запятой», `comma-separated values`) с тренировочными данными, собранными с примера целевого алгоритма `sin_ident.lua` из репозитория САПР NITTA.

Sid	Tag	Old_score	Is_terminal	Alt_bindings	Alt_refactorifgs	Alt_dataflow	PAAllow Data Flo	PAAlternative	PCritical	PNumber Of Bir	P
-62-58-54-51-3-48-17	BindDecisionView	2992	True	1	0	0	1	2	False	7	1
-62-58-54-51-3-48-17	ResolveDeadlock	1001	False	0	1	0					
-62-58-54-51-3-48-17	BindDecisionView	3372	False	0	0	0	0	1	False	18	1
-62-58-54-51-3-48-17	BindDecisionView	3462	False	1	0	0	0	1	False	9	1
-62-58-54-51-3-48-17	BindDecisionView	3472	False	2	0	0	0	1	False	8	1
-62-58-54-51-3-48-17	BindDecisionView	3482	False	3	0	0	0	1	False	7	1
-62-58-54-51-3-48-17	BindDecisionView	500	False	4	0	0	0	1	False	17	1
-62-58-54-51-3-48-17	BindDecisionView	500	False	5	0	0	0	1	False	16	1
-62-58-54-51-3-48-17	BindDecisionView	3492	False	6	0	0	0	1	False	6	1
-62-58-54-51-3-48-17	BindDecisionView	3402	False	7	0	0	0	1	False	15	1
-62-58-54-51-3-48-17	BreakLoopView	5000	False	8	0	0					
-62-58-54-51-3-48-17	BindDecisionView	3012	False	9	1	0	1	2	False	5	1
-62-58-54-51-3-48-17	BindDecisionView	500	False	10	1	0	0	1	False	14	1
-62-58-54-51-3-48-17	BreakLoopView	5000	False	11	1	0					
-62-58-54-51-3-48-17	BindDecisionView	2942	False	12	2	0	0	2	False	11	1
-62-58-54-51-3-48-17	BindDecisionView	500	False	13	2	0	0	1	False	13	1
-62-58-54-51-3-48-17	BindDecisionView	3432	False	14	2	0	0	1	False	12	1
-62-58-54-51-3-48-17	BreakLoopView	5000	False	15	2	0					
-62-58-54-51-3-48-17	ResolveDeadlock	1001	False	13	4	0					
-62-58-54-51-3-48-17	BindDecisionView	500	False	13	5	0	0	1	False	11	1
-62-58-54-51-3-48-17	BindDecisionView	3506	False	14	3	0	0	1	False	5	3

Рисунок 11 – Снимок экрана с демонстрацией CSV-файла с собранными тренировочными данными

Старым методом переработать это дерево синтеза в тренировочные данные невозможно из-за его слишком большого размера. Изображены тренировочные экземпляры, преимущественно – характеристики вершин. Пустые ячейки объясняются тем, что некоторые характеристики вершин могут отсутствовать для тех или иных типов действий в дереве синтеза.

На рисунке 12 изображен сам процесс сбора тренировочных данных и предусмотренное журналирование. С помощью библиотеки tqdm реализован индикатор выполнения, отображающий, среди прочего, ожидаемое оставшееся время и текущую производительность.

```

o devuser@42631918de77:/app$ python ml/synthesis/src/scripts/crawl_data_by_tree_sampling.py examples/pid.lua -n 1000
2023-05-27 21:57:39,156 INFO root Starting sampling of examples/pid.lua with 1000 samples
2023-05-27 21:57:39,156 DEBUG asyncio Using selector: EpollSelector
2023-05-27 21:57:39,156 DEBUG components.common.port_management Finding random free port...
2023-05-27 21:57:39,156 DEBUG components.common.port_management Finding out if port 47405 is in use...
2023-05-27 21:57:39,157 DEBUG components.common.port_management Port 47405 is free
2023-05-27 21:57:39,157 INFO components.common.port_management Found random free port: 47405
2023-05-27 21:57:39,157 INFO components.data_crawling.example_running Starting NITTA, cmd: stack exec nitta -- -p=47405 examples/pid.lua
2023-05-27 21:57:39,162 INFO components.data_crawling.example_running NITTA has been launched, PID 12192. Waiting for 2 secs.
[WARNING : NITTA.UI] WARNING: expected backend port: 8080 actual: 47405 (maybe you need regenerate API by nitta-api-gen)
Running NITTA server at http://localhost:47405 ...
2023-05-27 21:57:41,164 INFO components.data_crawling.example_running Retrieving tree root...
2023-05-27 21:57:41,169 INFO components.data_crawling.example_running Sampling tree (7 batches * 150 = 1000 samples >> 1 workers)...
Tree sampling: 30% ██████████ | 300/1000 [00:07<00:17, 39.49samples/s]

```

Рисунок 12 – Снимок экрана с демонстрацией процесса сбора тренировочных данных

3.4.2 Тестирование производительности параллельного сбора данных

Второй этап испытаний – тестирование различных реализованных способов параллелизации сбора данных, а также их влияния на производительность процесса. Это целесообразно сделать перед дальнейшим расширенным сбором тренировочных данных, чтобы максимизировать его скорость.

Тестирование производилось на Intel Core i5-12400F (6 физических, 12 логических ядер). Пример результатов тестирования при сборе данных с дерева синтеза целевого алгоритма `pid.lua` в течение 1000 итераций представлены в таблице 2. В ячейках – усреднённая производительность сбора (итераций / с), больше – лучше.

Таблица 2 – Результаты тестирования производительности сбора данных для `pid.lua`

		Количество копий САПР НИТТА		
		1	2	4
Режим параллелизации	Выключена	18	19	20
	Невытесняющая	39	39	40
	Мультипроцессная (3 процесса)	36	34	8

Выявлено, что наибольшего прироста удаётся добиться с помощью невытесняющей многозадачности (`async`, `await`). Мультипроцессный подход в лучшем случае не давал существенного прироста производительности, что может быть объяснено природой реализации: большая часть ожидания при исполнении приходится на сетевое взаимодействие, а не на нагружающие центральный процессор вычисления.

НИТТА производила основную часть вычислений и хорошо (с тем предположением, что бесполезная работа не производится) распределяла нагрузку по логическим ядрам процессора, о чём свидетельствовала их равномерная утилизация около 80%. Добавление второго работающего экземпляра САПР либо также значительно не улучшало производительность. Замедление происходило, предположительно, вследствие больших накладных расходов на гораздо более частую аппаратную смену контекста между потоками.

В ходе тестирования и отладки удалось дополнительно в несколько раз увеличить производительность решения с помощью переработки параллелизации на основе `async` и `await`. Функция, отвечающая за выполнение итерации, в начале сбора данных планировалась к исполнению (объект «`coroutine`») на событийном цикле (`event loop`)

асинхронной среды исполнения. С увеличением собираемого количества итераций росли накладные расходы на обслуживание запланированных функций в очереди. Когда таких функций становилось слишком много, сбор данных переставал работать. Решением стало уменьшение количества одновременно запланированных вызовов функций с помощью группировок итераций в «пачки» (batches). Функции планировались сразу такой пачкой с помощью `asuncio.gather` и выполнялись параллельно, не перегружая событийный цикл, что позволило увеличить производительность ещё в 2-3 раза на применяемом для тестирования ПК. В обсуждаемые результаты тестирования это уже отражено.

Кроме того, обнаружено, что оставлять дерево в оперативной памяти до тех пор, пока она не заполнена, тоже значительно ускоряет процесс за счёт кэширования уже полученной от НИТТА информации о вершинах дерева синтеза. Большая часть оперативной памяти в процессе сбора данных была занята самой САПР, а не обработчиком на Python, поэтому высвобождение включает, прежде всего, рестарт процесса самой САПР. Таким образом, решено высвобождать информацию из памяти только в условиях заполнения последней, чтобы дополнительно ускорить процесс сбора.

3.4.3 Сбор данных для подбора тренировочных целевых алгоритмов

Далее проведён предварительный сбор данных и оценка характеристик деревьев синтеза всех имеющихся примеров целевых алгоритмов, чтобы определить их пригодность к полноценному сбору для обучения модели. Использовались все доступные Lua файлы, в том числе написанные ранее в рамках настоящей работы для испытания новой стратегии обхода. При $n_samples=5000$ (относительно небольшое количество спусков) осуществлён сбор данных с деревьев синтеза, собраны n_p, n_v, r_c , а также вычислена доля меток ψ в тренировочных экземплярах, соответствующих неудачным результатам синтеза (они обратно распространялись в неизменном виде, использовалось значение -3, так как все оценки успешных результатов синтеза были больше 0). Собраны данные о более чем 2,7 миллионах уникальных вершин деревьев синтеза.

Выявлено, что на больших деревьях (например, дерево синтеза ПИД-регулятора) доля коллизий r_c не превышала 10 %. Стоит отметить, что 100% не соответствует полному покрытию дерева, r_c вследствие способа его определения может быть значительно больше 100%, а относительно полное покрытие дерева (более 85%) по экспериментальным данным достигается лишь при $r_c > 700\%$. Небольшая доля коллизий подтверждает относительную равномерность покрытия дерева синтеза процессом сбора данных и статистическую значимость собранных данных.

Обнаружено, что для многих деревьев собранные тренировочные данные будут не подходить для обучения, так как либо вообще не содержат метки неудачных результатов синтеза (обычно это небольшие деревья синтеза), либо не содержат в полученной выборке ни одной метки удачных результатов синтеза (обычно это самые большие деревья синтеза и сложные алгоритмы).

Необходимость реализации более продвинутого, неслучайного поиска удачных результатов в больших деревьях при сборе тренировочных данных – основное направление дальнейших исследований по усовершенствованию разработанного нового метода сбора данных. Это позволит иметь в тренировочных данных, собранных с деревьев сложных алгоритмов, как метки с успешными результатами синтеза, так и с неуспешными, что значительно повлияет на репрезентативность тренировочных данных.

На данный момент пришлось ограничиться при финальном сборе тренировочных данных использованием только тех примеров целевых алгоритмов, которые по результатам предварительного опроса демонстрировали наличие как меток успешных результатов синтеза, так и неуспешных: `pid.lua`, `teacup.lua`, `matrix-mult-1x3.lua`, `cyclic5.lua`, `cyclic4.lua`, `cyclic3.lua`, `vars.lua`, `constantFolding.lua`, `spi3.lua`, `sum.lua`. Среди используемых примеров присутствовали и некоторые с большими деревьями синтеза, но, как правило, чаще всего деревья были небольших или средних размеров (но всё ещё значительно больше, чем при сборе данных для первой версии прототипа).

3.4.4 Сбор тренировочных данных для обучения и испытания модели

Произведён финальный сбор тренировочных данных, тренировка новой модели и её сравнение с моделью из первой версии прототипа. Ожидается, что должна повыситься генерализуемость модели в частности и эффективность САПР в целом.

Для обучения модели успешно собран набор данных разработанным итеративным методом. Тренировочные экземпляры содержали информацию о более чем 3 миллионах уникальных вершин только с деревьев синтеза перечисленных выше алгоритмов. При этом сумма количества вершин в полных деревьях синтеза упомянутых алгоритмов оценивается как превосходящая собранное количество, как минимум, на один порядок. Количество итераций для различных примеров подобрано так, что в каждом дереве было посещено примерно 400 тыс. вершин. Доля меток неуспешных результатов синтеза в собранных данных была 49%, что допустимо.

На полученных данных натренирована модель МО для использования в качестве оценочной функции в процессе синтеза. Результат был назван «прототипом метода синтеза

на основе МО второй версии». Он был испытан с использованием разработанных ранее средств, запускающих САПР NITTA с различными CLI-аргументами, а затем измеряющих и сохраняющих различные характеристики результата. Менялась в данном случае используемая оценочная функция при синтезе, что стало возможным благодаря программной интеграции метода синтеза с МО в САПР, описанной в следующем разделе. Сравнивалась модель МО первой версии прототипа и второй с использованием различных стратегий обхода дерева синтеза, но основное внимание уделено новой стратегии обхода с $b = 1,2$, потому что, как отмечено ранее, такое значение параметра лучше подходит для синтеза с МО.

Обнаружено, что применение новой стратегии обхода повлияло на результаты синтеза всё же значительно, чем перетренировка текущей версии модели на новых собранных данных. Тем не менее, подтверждено умеренное статистически значимое увеличение суммарного количества успешных запусков процесса синтеза (не более, чем на 7%, однако для любых используемых стратегий обхода) и увеличение абсолютного количества успешно синтезируемых алгоритмов.

В частности, алгоритм `sin_ident.lua` не был ни разу удачно синтезирован моделью первой версии, модели второй версии это сделать удалось, причём дерево синтеза данного примера не включалось в тренировочные данные, что является проявлением улучшенной генерализуемости модели.

3.5 Выводы

Успешно спроектирован, реализован и испытан усовершенствованный итеративный метод сбора тренировочных данных. Разработанный исходный код и сводная таблица результатов испытаний опубликованы в открытый доступ в репозитории САПР спецвычислителей NITTA [30, 37].

Испытания подтвердили соответствие нового метода всем сформулированным ранее требованиям, позволили выявить узкие места и дополнительно ускорить процесс сбора данных. Осуществлён повторный сбор данных и тренировка модели, применяемой в качестве оценочной функции в процессе синтеза, и положительное влияние новых тренировочных данных на генерализуемость модели и, следовательно, на эффективность САПР, было подтверждено.

В качестве направлений дальнейшего развития средства можно отметить, прежде всего, изучение способов добиться более равномерной репрезентации успешных и

неуспешных результатов синтеза в тренировочных данных, собранных с больших деревьев синтеза сложных целевых алгоритмов. Кроме того, требует доработки средство аппроксимации доли покрытия дерева после по результатам сбора данных.

4 ВСТРАИВАНИЕ СИНТЕЗА НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ В NITTA

Для проведения испытаний разработанных средств и для развития их практической применимости необходимо программно интегрировать прототип метода синтеза на основе МО в поток исполнения САПР NITTA, то есть сделать доступной модель МО в качестве оценочной функции внутри самой САПР. Это сложная и объёмная задача, поэтому её описание вынесено в отдельный раздел.

4.1 Исследование технических аспектов синтеза в NITTA

Прежде всего, необходимо на уровне программного кода изучить существующую реализацию процесса синтеза в САПР NITTA, чтобы учесть её особенности при проектировании как можно раньше, минимизировав технические риски, которые могут привести к значительному увеличению трудозатрат в дальнейшем.

В проекте NITTA код основной части САПР написан на функциональном языке Haskell. Используется средство для сборки и управления проектами Stack. Для работы решено использовать единое окружение разработчика (Docker-контейнер), создание которого описано в следующем разделе. Использование контейнеризованного окружения упростит настройку инструментов, а также позволит испытать созданное средство и собрать информацию для дальнейшей его доработки. Модули, связанные с МО и реализацией прототипа, написаны на языке Python.

Работа производится с исходным кодом, доступным в публичном репозитории САПР NITTA [30]. За реализацию релевантного функционала отвечает прежде всего набор модулей в директории `src/NITTA/Synthesis`. Точками входа к этому функционалу являются модули `NITTA.Synthesis` (неинтерактивный синтез), `NITTA.UiBackend.REST` (интерактивный синтез посредством веб-интерфейса NITTA) и, транзитивно, модуль `Main`, где по необходимости происходит запуск сервера веб-API (application programming interface, API) или вызов релевантных функций для синтеза целевой системы в неинтерактивном режиме. Кроме того, имеются зависимости в автоматизированных тестах. Эти факторы усложняют изменение типов функций, связанных с синтезом.

Модуль NITTA.Synthesis.Explore отвечает за сборку дерева синтеза и продвижению по нему вглубь, NITTA.Synthesis.Method реализует различные стратегии и механизмы обхода дерева.

Основной целью данного анализа являлось определение контекста, в котором происходит оценка вершин. Большая часть действий происходит в монаде IO, что позволит осуществить в коде необходимые взаимодействия с «внешним миром», но непосредственно оценка вершин происходит в контексте монады STM (Software Transactional Memory, программная транзакционная память), где осуществление операций ввода-вывода с помощью монады IO реализовывать нецелесообразно, так как для этого нет явной поддержки и это может приводить к проблемам. Например, действия в монаде STM могут повторяться неопределённое количество раз в случае конфликтов транзакций, что при наличии внешних взаимодействий, во-первых, неэффективно, а во-вторых, может привести к ошибкам в гипотетическом случае отсутствия идемпотентности операций взаимодействия.

Чтобы обойти указанную проблему, решено вычислять оценки с помощью МО после вычисления «классической» оценки и выше по стеку вызовов. Таким образом, у каждой вершины появится несколько именуемых по-разному оценок, которые можно сохранить в ассоциативном массиве. Другие подходы требуют значительной переработки кода и более глубокого понимания проекта, а также не относятся напрямую к теме работы, поэтому вынесены за рамки настоящей диссертации.

4.2 Программная интеграция прототипа и NITTA

После сбора необходимой информации стало возможно провести анализ различных технических способов достижения поставленной задачи, а также детальное проектирование конечного решения.

4.2.1 Поиск способов интеграции прототипа и кода NITTA

Необходимо найти способы интеграции кода на языке Haskell, отвечающего за синтез, и кода на языке Python, вычисляющего оценки вершин с помощью МО и библиотеки TensorFlow [38]. Рассмотрены следующие возможные подходы:

- 1) динамически подключаемая библиотека TensorFlow на C (TensorFlow C API) [39] с использованием библиотеки для FFI-вызовов (Foreign Function Interface) TensorFlow из Haskell [40];
- 2) динамически подключаемая библиотека Python (Python C API) с возможностью запуска встроенного в другой процесс интерпретатора [41] с использованием библиотеки для FFI-вызовов Python из Haskell [42];
- 3) вызовы к серверу TensorFlow Serving [43] по протоколу HTTP (Hypertext Transfer Protocol);
- 4) вызовы к собственному серверу на Python по протоколу HTTP;
- 5) прочие механизмы межпроцессного взаимодействия (inter-process communication, IPC) с отдельным приложением на Python: собственные протоколы поверх стандартных потоков ввода-вывода или сокетов, удаленный вызов процедур (remote procedure call, RPC).

4.2.2 Сравнительный анализ способов интеграции

Каждый указанный способ обладает набором особенностей и характеристик. Предлагается проанализировать их, сопоставить с целями и выбрать, какой будет использоваться в конечном решении.

Все представленные способы можно разделить на две группы: зависящие от TensorFlow и способные работать с любой библиотекой МО. В ходе дальнейшего развития прототипа метода синтеза на основе МО могут быть применены и другие библиотеки, реализующие модели отличных видов и архитектур, удобной поддержки которых в TensorFlow нет, поэтому предпочтительным сочтено интеграционное решение, не зависящее от библиотеки МО.

Кроме того, способы можно разделить и на две другие группы: использующие динамически подключаемые библиотеки и использующие подпроцессы. Способы первой группы предполагают обращение к компонентам, связанным с МО, программными вызовами в рамках одного процесса и адресного пространства к динамическим библиотекам, которые должны быть установлены в системе пользователя. Способы второй группы для обработки аналогичных запросов запускают подпроцесс, используя различные механизмы IPC для взаимодействия: протокол HTTP, стандартные потоки, сокет, RPC.

Способы с динамическими библиотеками, как правило, производительнее, но усложняют сборку проекта, усиливают связность компонентов, предполагают обязательность библиотек (NITTA не сможет запуститься, если МО компоненты

отсутствуют, даже если не планируется их использовать), а также сложнее в переносе между различными операционными системами из-за потенциально несовместимого двоичного интерфейса (application binary interface, ABI) при отличающихся версиях компонентов. Это было выяснено в процессе непосредственных попыток сборки проекта с подключением и вызовами libpython3.8-dev (Python C API). Удалось это успешно сделать, но решение отличалось сложностью и затруднённой переносимостью на другие платформы.

Способы с подпроцессами же, с другой стороны, подразумевают необходимость реализации управления ими и менее производительны, но не имеют описанных выше проблем в той же степени. Программную реализацию для таких решений организовать гораздо проще ввиду наличия большого количества продвинутых библиотек. Кроме того, необязательность загрузки МО компонентов на данном этапе их развития кажется особенно важной возможностью. Следовательно, предпочтительными решено считать способы с подпроцессами.

Краткие итоги сравнительного анализа представлены в таблице 3. Способы, помимо упомянутых выше классов, также классифицированы по другим категориям, влияющим на принятие решения:

- *сложности* реализации и сборки проекта, включая вопрос кроссплатформенности и необходимости дополнительных действий со стороны пользователя;
- *относительной производительности*, объясняемой, например, разницей в дополнительных затратах на сериализацию и передачу данных;
- *модульности* – возможности запускать NITTA без компонентов МО.

Таблица 3 – Краткие итоги сравнительного анализа способов интеграции

Способ Характеристика	TensorFlow C API	Python C API	TensorFlow Serving	Веб-сервер на Python	Подпроцесс Python и IPC
Зависимость от TensorFlow	+	–	±	–	–
Принцип	Библиотека	Библиотека	Подпроцесс	Подпроцесс	Подпроцесс
Сложность	Высокая	Высокая	Средняя	Низкая	Средняя
Производительность	Высокая	Средняя	Средняя	Низкая	Низкая/ Средняя
Модульность	Нет	Нет	Да	Да	Да

4.2.3 Проектирование конечного решения

На основе сравнительного анализа приняты следующие решения:

- отказ от способа TensorFlow C API из-за зависимости от TensorFlow, высокой сложности реализации и несоответствия критерию модульности;
- отказ от способа Python C API из-за высокой сложности реализации и несоответствия критерию модульности;
- отказ от использования подпроцесса Python и собственных механизмов IPC, так как увеличение сложности реализации видится несопоставимым с получаемыми преимуществами относительно других подходов с подпроцессом (потенциальный прирост производительности);
- отказ от использования TensorFlow Serving, так как сложнее в реализации и требует значительной переработки уже написанных модулей, хоть способ и видится относительно подходящей альтернативой выбранному подходу, так как теоретически поддерживает сторонние модели и представляет собой готовый оптимизированный веб-сервер под решаемую задачу;

Сделан выбор в пользу собственного веб-сервера на Python с обращением к нему из НИТТА по протоколу HTTP в качестве финального используемого решения. Такой подход позволит относительно просто заменить библиотеку МО в будущем или даже совместить несколько за одним интерфейсом, сделает возможным запуск НИТТА без МО, значительно проще альтернатив в разработке и отладке, а также позволит переиспользовать почти в готовом виде существующие программные модули. Кроме того, подход даёт возможность расширить функционал компонентов МО в будущем. Например, он позволяет реализовать запрос списка доступных моделей для оценки вершин, что можно будет отобразить и в веб-интерфейсе НИТТА, а также может позволить менять используемую модель «на ходу» при интерактивном синтезе. Из недостатков – необходимость реализовывать управление подпроцессом и дополнительные затраты вычислительных мощностей на сериализацию и десериализацию данных, а также сетевое взаимодействие.

Для реализации сервера на Python (выбранное техническое название компонента – «ML Backend», «сервер МО») используется фреймворк FastAPI, так как он отличается простотой использования, производительностью и при этом достаточной функциональностью [44]. Предполагается, что сервер будет реализовывать веб-интерфейс для оценки вершин, а НИТТА в процессе синтеза будет осуществлять синхронные сетевые вызовы к этому интерфейсу. Так как сервер запущен локально, то скорость сетевого взаимодействия должна быть пренебрежимой.

В интерфейсе должен быть реализован, прежде всего, метод для оценки вершин, который принимает список наборов входных данных и возвращает оценки для каждого набора. Таким образом, в одном запросе может быть оценено сразу множество вершин. Такой подход позволит сэкономить количество вызовов.

Кроме того, структура набора входных данных спроектирована так, чтобы она не зависела от модели. Другими словами, она содержит достаточно информации для оценки вершин любой моделью. Это позволит избежать разветвления кода сбора данных на стороне Haskell в зависимости от модели, с которой он взаимодействует в данный момент. Так ответственность за форматирование входных данных модели из кода Haskell перенесена в код сервера МО. В частности, это позволило технически несложно при испытаниях провести сравнение модели МО первой версии и второй (у них разное количество используемых входных характеристик). Кроме того, такое решение необходимо, если в будущем появится более одной модели для оценки вершин (ожидается, что так и будет). Некоторые модели могут быть экспериментальными, некоторые могут лучше подходить для одного типа синтезируемых специализированных вычислителей, некоторые — для другого.

Для исключения конфликтов с другими программами для запуска сервера МО используется случайный свободный порт. Информация о ссылке для доступа к серверу записывается при его старте в рабочую директорию в файл `.ml_backend_base_url`, откуда код NITTA сможет её прочитать. Так процесс САПР получит информацию, во-первых, что сервер МО успешно запущен, и, во-вторых, на каком порту сервер работает. Кроме того, это позволит запускать сервер МО отдельно от NITTA, что значительно ускорит работу системы при частых рестартах процесса NITTA.

Для обеспечения обратной совместимости важно было сохранить таким же поведение NITTA по умолчанию, поэтому функционал синтеза на основе МО включается с помощью добавленного к главному исполняемому файлу САПР CLI-аргумента `--score=<имя оценочной функции>`. Передав имя оценочной функции вида `ml_<имя модели МО>` пользователь сможет включить оценку вершин с помощью МО и указать имя модели, которую необходимо использовать. Если пользователь уже запустил сервер МО отдельно, то будет использоваться именно он, а если нет, то предполагаются автоматические запуск и остановка сервера МО. Для обращения к интерфейсу сервера МО переиспользуются структуры данных из NITTA REST API, так как существующие модули МО работают с этими структурами и зависят от них.

4.3 Реализация интеграции прототипа и процесса синтеза в NITTA

В соответствии со сформированным проектом готового решения реализован сервер МО с автоматически генерируемой документацией, а также необходимые программные модули на языке Haskell. Успешно реализован вызов сервера МО и вычисление оценок для новых посещаемых вершин с его помощью.

Использован шаблон Bracket [45], который позволит завершить автоматически запущенный подпроцесс сервера МО при завершении процесса NITTA. Автоматический запуск подпроцесса при этом производится «лениво», то есть только по требованию, и только если не найден файл, сообщивший ссылку на уже запущенный вручную сервер МО (рисунки 13, 14).

```
[DEBUG : NITTA.Synthesis] reading ML backend base URL from file
[DEBUG : NITTA.Synthesis] ML backend server base URL was found (http://localhost:8000), skipping server startup
"Prediction result: Just 1.3"
"Prediction result: Just 1.3"
"Prediction result: Just 1.3"
[DEBUG : NITTA.Synthesis] ML backend server was not started automatically, so nothing to stop
```

Рисунок 13 – Автоматический запуск подпроцесса не производится, так как найден уже запущенный сервер МО

```
registering library for nitto-0.0.0.1...
[DEBUG : NITTA.Synthesis] reading ML backend base URL from file
[DEBUG : NITTA.Synthesis] failed to read ML backend base URL from file
[INFO : NITTA.Synthesis] Starting ML backend server, expected executable: /app/nitita-mlbackend
[DEBUG : NITTA.Synthesis] waiting 30 second(s) more until ML backend server base URL is available...
[DEBUG : NITTA.Synthesis] reading ML backend base URL from file
[DEBUG : NITTA.Synthesis] failed to read ML backend base URL from file
[DEBUG : NITTA.Synthesis] waiting 27 second(s) more until ML backend server base URL is available...
[DEBUG : NITTA.Synthesis] reading ML backend base URL from file
[DEBUG : NITTA.Synthesis] failed to read ML backend base URL from file
[DEBUG : NITTA.Synthesis] waiting 24 second(s) more until ML backend server base URL is available...
2023-03-27 00:27:59.352538: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcudart.so.11.0
2023-03-27 00:28:00,377 DEBUG components.common.port_management Finding random free port...
2023-03-27 00:28:00,377 DEBUG components.common.port_management Finding out if port 35103 is in use...
2023-03-27 00:28:00,377 DEBUG components.common.port_management Port 35103 is free
2023-03-27 00:28:00,377 INFO mlbackend.backend_base_url_file Writing base URL (http://127.0.0.1:35103) to file (/app/.ml_backend_base_url).
2023-03-27 00:28:00,378 INFO root Starting ML backend server on port 35103
INFO: Started server process [19981]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:35103 (Press CTRL+C to quit)
2023-03-27 00:28:01.927502: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcuda.so.1
INFO: 127.0.0.1:58218 - "POST /models/production/score HTTP/1.1" 200 OK
"Prediction result: Just 1.3"
2023-03-27 00:28:02,252 DEBUG mlbackend.models_facade Getting model production
INFO: 127.0.0.1:58218 - "POST /models/production/score HTTP/1.1" 200 OK
"Prediction result: Just 1.3"
2023-03-27 00:28:02,254 DEBUG mlbackend.models_facade Getting model production
INFO: 127.0.0.1:58218 - "POST /models/production/score HTTP/1.1" 200 OK
"Prediction result: Just 1.3"
[INFO : NITTA.Synthesis] Stopping automatically started ML backend server
INFO: Shutting down
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
INFO: Finished server process [19981]
2023-03-27 00:28:02,369 INFO mlbackend.backend_base_url_file Exiting, so removing /app/.ml_backend_base_url.
```

Рисунок 14 – Запущенный сервер МО не найден, производится автоматический запуск и остановка сервера МО при завершении родительского процесса

Для форматирования тела запроса использована уже применённая в проекте библиотека Aeson и реализованные структуры данных в модуле ViewHelper, а именно –

NodeView. Однако, это привело к циклическим зависимостям между модулями (рисунок 15). В вершинах – модули Haskell, на ребрах – зависимости между ними.

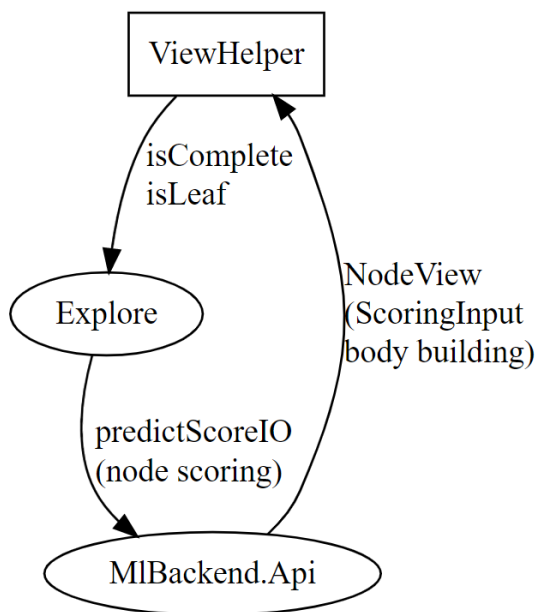


Рисунок 15 – Циклические зависимости (упрощенно)

Проанализированы возможные варианты решения проблемы. Вариант 1 – обобщить новую структуру данных ScoringInput, используемую для отправки данных о вершинах на оценку. Это сделает ее независимой от NodeView. Иллюстрация приведена в листинге 1.

Листинг 1 – иллюстрация генерализации структуры данных ScoringInput

```

data ScoringInput = ScoringInput{ nodes :: NodeView } -- old
data ScoringInput nv = ScoringInput{ nodes :: nv } -- new
  
```

Этот вариант семантически неверный, так как на данный момент сервер МО во входящей структуре данных ScoringInput ожидает только тип NodeView, что сделано специально с целью вынести ответственность за сбор и преобразование данных для подачи на вход модели из кода НИТТА в код сервера МО. Если генерализовать тип, то в коде НИТТА появится возможность добавлять логику форматирования данных в зависимости от того, какая модель сейчас вызывается, что сочтено нежелательным.

Вариант 2 – написать собственную, отдельную структуру данных. Модуль ViewHelper изначально принадлежал к верхнему уровню (представления), поэтому зависимость снизу вверх действительно кажется необоснованной: необходимо реализовывать отдельный уровень представления и отдельную структуру данных. В ходе попытки такой реализации выявлено, что, во-первых, результат неудобен в использовании

с программной точки зрения (повторная реализация большого блока функционала по форматированию вершины дерева синтеза, что фактически есть копирование и вставка), а, во-вторых, лишён смысла, так как новая структура имеет фактически ту же роль, что и NodeView. У них нет различий, одинаковая ответственность, независимые друг от друга изменения в них маловероятны. Следовательно, дублирование большого объёма кода нежелательно.

Вариант 3 – вынести оценку вершин из модуля Explore. К сожалению, у этого функционала много зависимостей в модулях REST и Method, поэтому такой подход приведёт к большому количеству изменений. Кроме того, такая переработка идёт вразрез со смыслом и логикой текущей реализации, где оценка вершин производится в Explore, а результат записывается в структуру данных SynthesisDecision.

В поисках подходящего варианта была более подробно проанализирована структура зависимостей (рисунок 16). Круглыми вершинами показаны модули директории Synthesis, квадратными – UiBackend. Обнаружено, что в модуле Explore объявлены функции isComplete и isLeaf, которые используются только в других модулях (на рисунке – красным)

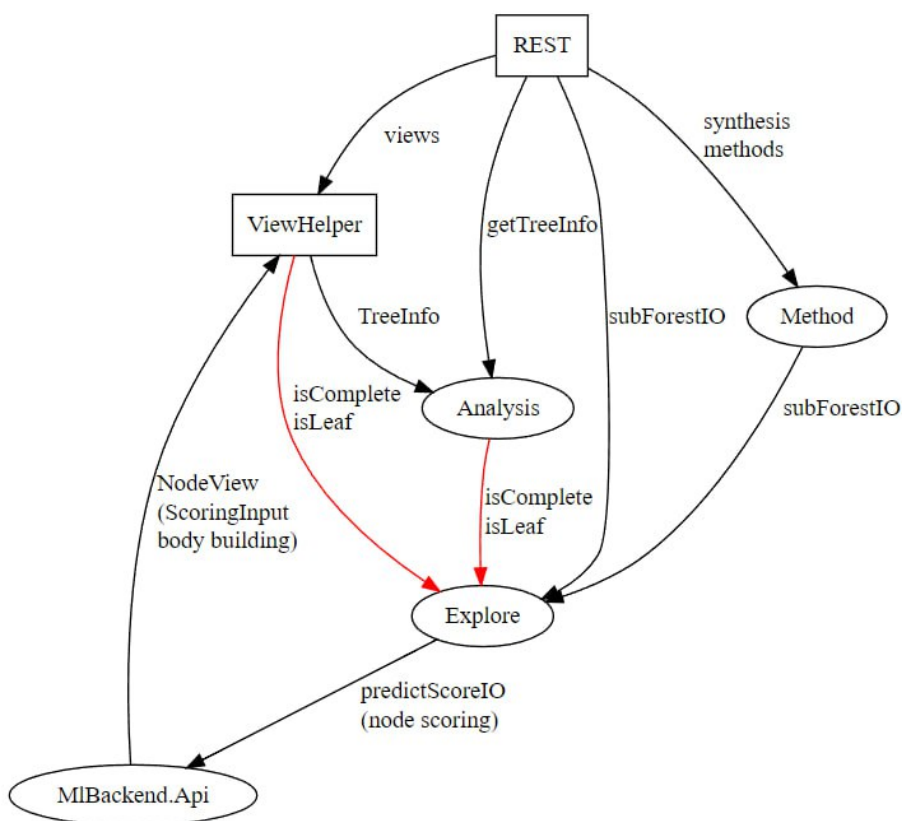


Рисунок 16 – Подробная структура дерева зависимостей

Так найден ещё один вариант решения проблемы – перенести isComplete и isLeaf из модуля Explore в модуль Analysis. Это обоснованно следующими соображениями:

- ViewHelper зависит, на самом деле, только от этих функций;
- функции мелкие и вспомогательные (изменение кода будет небольшим);
- по смыслу функции действительно скорее анализируют вершины дерева, а не шагают по нему вглубь;
- при применении такой переработки дерево зависимостей получается логичнее и стройнее, а циклические зависимости устраняются (рисунок 17).

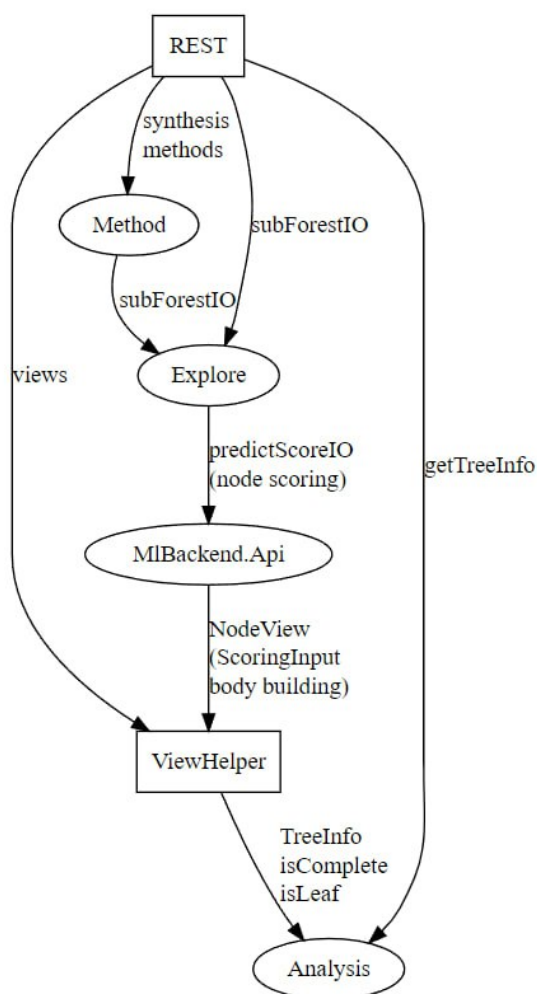


Рисунок 17 – Подробное дерево зависимостей после переработки

В итоге, как самое обоснованное и простое, выбрано именно такое решение проблемы с циклическими зависимостями.

4.4 Тестирование, отладка и документирование реализации

Полученная реализация была протестирована с использованием различных целевых алгоритмов синтезируемого специализированного вычислителя (рисунок 18).

Binding												
sid	Z(d)	description	crit	lock	wave	outputs	alt	rest	newDF	newBind	inputs	
0 >	0.3899	fram1 ← loop(180.000000, temp_cup^1#0) = temp_cup^0#0 = temp_cup^0#1 = temp_cup^0#0	true	false	1	3	2	0	0	0	1	①
1 >	0.3899	fram2 ← loop(180.000000, temp_cup^1#0) = temp_cup^0#0 = temp_cup^0#1 = temp_cup^0#0	true	false	1	3	2	0	0	0	1	①
2 >	0.2498	fram1 ← loop(0.000000, time^1#0) = time^0#0 = time^0#1	true	false	1	2	2	0	0	0	0	①
3 >	0.2498	fram2 ← loop(0.000000, time^1#0) = time^0#0 = time^0#1	true	false	1	2	2	0	0	0	0	①
4 >	0.7863	accum ← temp_room^0#0 - temp_cup^0#1 = acc^0#0	false	false	0	1	1	0	2	1	0	①
5 >	0.7863	accum ← time^0#1 + time_step^0#0 = time^1#0	false	false	0	1	1	0	2	1	0	①
6 >	0.19123	spi ← send(temp_cup^0#0)	false	false	0	0	1	0	1	0	0	①
7 >	0.19123	spi ← send(time^0#0)	false	false	0	0	1	0	1	0	0	①
8 >	0.4592	fram1 ← const(0.000000) = time_step^0#0 = time_step^0#1	false	false	0	2	2	0	0	0	1	①
9 >	0.4592	fram2 ← const(0.000000) = time_step^0#0 = time_step^0#1	false	false	0	2	2	0	0	0	1	①
10 >	0.3593	fram1 ← const(70.000000) = temp_room^0#0	false	false	0	1	2	0	0	0	1	①
11 >	0.3593	fram2 ← const(70.000000) = temp_room^0#0	false	false	0	1	2	0	0	0	1	①
12 >	0.3593	fram1 ← const(10.000000) = temp_ch^0#0	false	false	0	1	2	0	0	0	1	①
13 >	0.3593	fram2 ← const(10.000000) = temp_ch^0#0	false	false	0	1	2	0	0	0	1	①

Рисунок 18 – Снимок части пользовательского веб-интерфейса NITTA, демонстрирующий использование системой оценок вершин от модели МО (столбец $Z(d)$)

Производительность синтеза ожидаемо была значительно хуже решения без сетевого взаимодействия, но всё ещё в пределах допустимого.

В ходе тестирования выявлено, что модель гораздо чаще склонна давать отрицательные оценки вершинам, а в реализации NITTA такие вершины не учитываются при синтезе, так как «классические» отрицательные оценки имеют другой смысл. На данный момент проблема решена сдвигом ответов модели на +20 единиц. Возможно, решение стоит пересмотреть в будущем, унифицировав поддержку различных типов оценок.

В автоматизированный «smoke» тест функционала, связанного с МО, добавлена высокоуровневая проверка возможности NITTA использовать оценки вершин от сервера МО при синтезе с самостоятельным запуском и остановкой сервера.

Сервер МО снабжён автоматически генерируемой документацией по его API, которая была заполнена: дано описание полям данных и методам, определены примеры. Кроме того, добавлен более функциональный интерфейс для просмотра этой документации (рисунок 19).

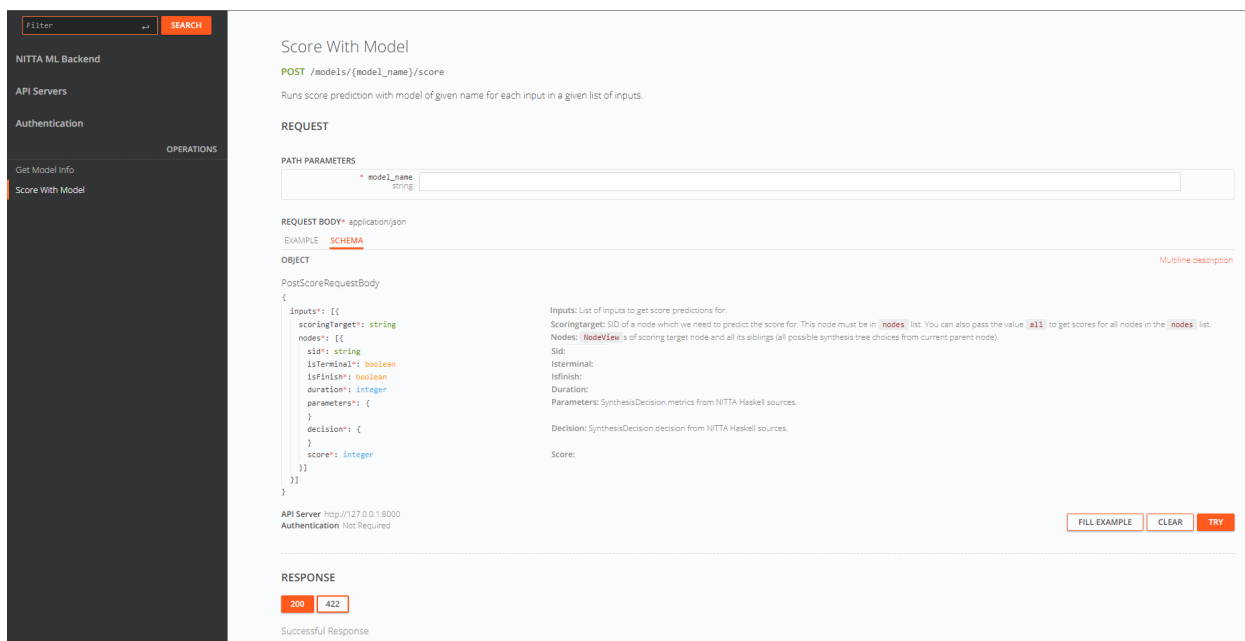


Рисунок 19 – документация API разработанного сервера МО

4.5 Выводы

Произведен тщательный анализ механизма работы процесса синтеза в реализации САПР NITTA и его особенностей. Найдены и проанализированы способы программной интеграции разработанных модулей МО на языке Python и кода САПР на языке Haskell. Сделан мотивированный выбор используемого способа. На основе полученной информации детально спроектировано и реализовано решение поставленной задачи. Реализация была задокументирована, разработаны соответствующие автоматические тесты, обнаруженные при тестировании проблемы были исправлены. Выявлены и обозначены возможные направления развития разработанных средств.

Удалось успешно решить поставленную задачу и использовать прототип в интегрированном виде при всех испытаниях и экспериментах, проведенных в настоящей работе.

5 КОНТЕЙНЕРИЗАЦИЯ РАБОЧЕГО ОКРУЖЕНИЯ

В ходе выполнения работы выявлен ряд технических трудностей, связанных со сложностью окружения разработчика, что в значительной степени негативно влияет на опыт разработки (developer experience, DX) САПР НИТТА и прототипа метода синтеза на основе МО. Первоначальная настройка может занимать до нескольких часов на каждой машине, так как окружение включает в себя множество различных средств.

Упростить их установку и настройку могут помочь сборочные скрипты, автоматизирующие все необходимые действия, однако в простейшем варианте разрабатывать такие скрипты необходимо для каждой платформы. Кроме того, используемые наборы инструментов могут в такой конфигурации значительно отличаться, что может усложнить взаимодействие разработчиков в проекте и негативно сказаться на DX. Например, наладку работы связанных с Haskell расширений интегрированной среды разработки необходимо будет производить заново для каждой новой IDE.

Сделать сборочные скрипты платформонезависимыми, то есть обеспечить их одинаковую работу на разных операционных системах (прежде всего, на Windows, macOS и дистрибутивах Linux), помогут средства контейнерной виртуализации, такие как Docker. Виртуализация позволит сделать окружение разработки единым для всех, а контейнеризация обеспечит отсутствие избыточности, связанной с использованием в качестве рабочего окружения проекта полноценного образа виртуальной машины (VM), а также упростит интеграцию с хостовой операционной системой (ОС).

Удобным для разработчика подобное решение можно сделать благодаря инструментам удалённой разработки, которые отделяют интерфейс IDE от среды исполнения кода, организуя общение этих двух компонентов по сети. Такое разделение позволит редактировать код на локальной машине, а исполнять, отлаживать и тестировать его на «удалённой», то есть в случае предлагаемого решения – в контейнере.

5.1 Определение списка необходимых средств

Прежде всего необходимо определить состав создаваемого окружения для разработки проекта НИТТА, а также список средств, необходимых для реализации такого контейнеризованного окружения. Окружение разработчика проекта НИТТА включает в себя следующие компоненты:

- связанные с основной частью кода на языке Haskell:
 - а) средство для сборки и управления проектами Stack;
 - б) компилятор (Glasgow Haskell Compiler, GHC);
 - в) симулятор языка Icarus Verilog (iverilog);
 - г) средства для форматирования и аудита кода: hlint, fourmolu;
 - д) скомпилированные зависимости;
- среду для разработки веб-интерфейса:
 - а) среду исполнения Node;
 - б) пакетный менеджер Yarn;
 - в) средства для форматирования и аудита кода: Prettier, ESLint;
 - г) зависимости (React, TypeScript, сборщик проекта);
- среду для МО:
 - а) интерпретатор Python 3;
 - б) библиотеку Tensorflow (её размер – более 1 ГБ);
 - в) NVIDIA CUDA Toolkit и библиотека cuDNN для тренировки моделей с использованием графических процессоров (Graphics Processing Unit, GPU);
 - г) Jupyter Notebook для интерактивного исполнения кода.

Конечному пользователю окружения будет необходимо установить только Docker с некоторыми вспомогательными инструментами в зависимости от платформы:

- на Windows – Docker Desktop и WSL 2;
- на Linux – Docker и NVIDIA Container Toolkit для поддержки GPU в контейнере [46];
- на macOS – Docker Desktop for Mac.

Значительной части средств удалённой разработки для работы в контейнере необходимо наличие Secure Shell (SSH) сервера в удалённом окружении [47, 48], поэтому в окружение будет добавлен и SSH-сервер, а также автоматическая генерация новой пары ключей для удобства подключения.

Для этого написан дополнительный скрипт – точка входа (entrypoint). Он будет исполняться при каждом запуске контейнера. Кроме того, он используется для запуска сервисов (например, Jupyter Notebook), вывода отладочной информации (количество доступных GPU) и приветственного сообщения с инструкцией по подключению к контейнеру или сервисам.

5.2 Реализация и отладка сборочных скриптов

Разработка и отладка производились на Docker Desktop for Windows с WSL 2. В качестве первой ступени реализован Dockerfile, устанавливающий основные компоненты (Stack, GHC, Node, Yarn, Python), загружающий и осуществляющий сборку зависимостей.

Поддержка GPU в Tensorflow требовала большого (более 8 ГБ) дополнительного пакета зависимостей, поэтому с помощью многоступенчатой сборки выделено два собираемых образа: один с поддержкой GPU для тех разработчиков, кому она понадобится, и другой – без поддержки, чтобы сэкономить время и пространство на устройствах остальных разработчиков.

Файлы в хостовой операционной системе (ОС) и контейнере синхронизируются с помощью Docker Bind Mounts [49]. Это позволяет моментально обмениваться изменениями в файлах между окружениями. Решены проблемы с тем, что у файлов были неправильные права доступа и владельцы с несуществующими идентификаторами (они были в хостовой ОС, но не в контейнере), посредством создания в контейнере пользователя с нужными идентификаторами, совпадающими с хостовой ОС, от имени которого предполагается разработка.

При разработке окружения решены и ряд других проблем. Например, добавление нового пользователя в контейнере привело к необходимости сборки зависимостей блока кода на Haskell от имени нового пользователя, а не от имени пользователя root, как было раньше, потому что иначе собранные зависимости не обнаружит Stack (он работает в отдельной директории для каждого пользователя и попытки это переконфигурировать ожидаемо привели к проблемам с правами доступа к файлам пользователя root).

Кроме того, при установке средств hlint и fourmolu некоторые зависимости не могли скомпилироваться. Причина по ошибкам в выводе Stack неочевидна (ненулевой код возврата). Помог понять проблему запуск сборки проблемных пакетов вручную с более подробным отладочным выводом ошибок, что указало на отсутствие некоторых динамических библиотек. Поиск и добавление инструкций на установку релевантных системных пакетов в образ решили проблему.

По вопросу конфигурации сети рассмотрена возможность использовать специальный ключ при запуске контейнера, отключающий разделение сетей хостовой машины и контейнера (`--network=host`), но он не поддерживается в Docker Desktop на Windows (и, потенциально, на macOS), в связи с чем известные базовые порты предлагается

отображать при запуске контейнера (SSH-сервер, Jupyter Notebook), а остальные предлагать отображать с помощью SSH по запросу пользователя.

5.3 Организация поддержки графических процессоров

Несмотря на сложности, удалось успешно организовать работу GPU в контейнере. На Windows для этого необходимо заставить, прежде всего, работать GPU в WSL, так как контейнеры запускаются именно в этом окружении. Выбрана WSL 2, так как она поддерживает GPU от NVIDIA по умолчанию [50, 51], без каких-либо дополнительных действий. Это подтверждается с помощью вызова средства `nvidia-smi`.

Следующий этап – добавление поддержки непосредственно в Docker-контейнере, для чего компанией NVIDIA был разработан NVIDIA Container Toolkit [46]. Он совместим с Docker Desktop на Windows и встроен по умолчанию в актуальные драйвера GPU для хостовой машины, если при запуске контейнера добавлен CLI-аргумент `-gpus=all`. На Linux необходимо устанавливать Toolkit вручную, но есть готовые пакеты для пакетных менеджеров, что сводит установку к выполнению одной команды. На macOS, к сожалению, поддержка использования GPU в Docker на момент написания отсутствует.

Требовали добавления в образ и связанные с GPU зависимости. В сборочный Dockerfile включена установка Tensorflow-GPU, CUDA Toolkit и библиотека cuDNN. Для обеспечения совместимости их версий написана соответствующая проверка при сборке образа.

CUDA Toolkit был установлен в соответствии с документацией [52] из репозитория пакетов. Несмотря на сложность такого способа, это выглядит единственным вариантом решения, так как необходимо было устанавливать версии 11.x, не распространяемые через Python Package Index, как современные. Установка cuDNN по документации была неочевидной, но найден соответствующий пакет в том же репозитории, где был CUDA Toolkit, поэтому cuDNN установлен аналогично.

В завершение, в точку входа добавлена проверка на количество доступных GPU. Она успешно исполнялась и выводила реальное количество установленных графических процессоров (1 при тестировании).

5.4 Интеграция решения в проект и его апробация

Следующим шагом стала интеграция разработанного решения в проект, изучение и документирование его возможностей, а также более тщательное тестирование.

5.4.1 Подбор и проверка средств удаленной разработки в контейнере

Удобные интерфейсы для работы с кодом и отладки в значительной степени влияют на DX, поэтому для организации эффективного контейнеризованного окружения разработчика важно предложить готовые решения, отвечающие на этот запрос.

Прежде всего, так как возможен доступ к контейнеру с помощью терминала или `docker attach`, будут работать любые терминальные редакторы кода и средства разработки, такие как `vim`, `emacs`, `gdb`, и так далее.

Кроме того, относительно несложно настраиваются средства разработки с веб-интерфейсом, такие как Jupyter Notebook (решает часть задач при разработке модулей МО) и Visual Studio Code Server [53]. Последний в рамках тестирования был установлен и запущен в контейнере. Для доступа к нему использовался порт, отображённый с помощью SSH. Такой вариант отлично подойдёт в определённых сценариях, но разрабатывать в отдельном окне чаще всего показалось более удобным, чем в браузере.

Рассмотрены полноценные IDE с разработкой на удалённой машине по SSH. Удалось использовать расширение для удалённой разработки в Visual Studio Code, снимок экрана изображен на рисунке 20. Порты запущенных приложений автоматически отображались в хостовую ОС, а система контроля версий работала, как обычно, за исключением необходимости реализовать обходной путь для передачи GPG-подписей коммитов. Доступен терминал для облегчения работы со средствами сборки. На этом этапе также в скрипт сборки образа добавлены команды для обновления `git`, чтобы исправить CVE-2022-24765, о которой IDE сообщала. Поддержка данной IDE особенно важна для САПР NITTA, так как она популярна среди разработчиков проекта.

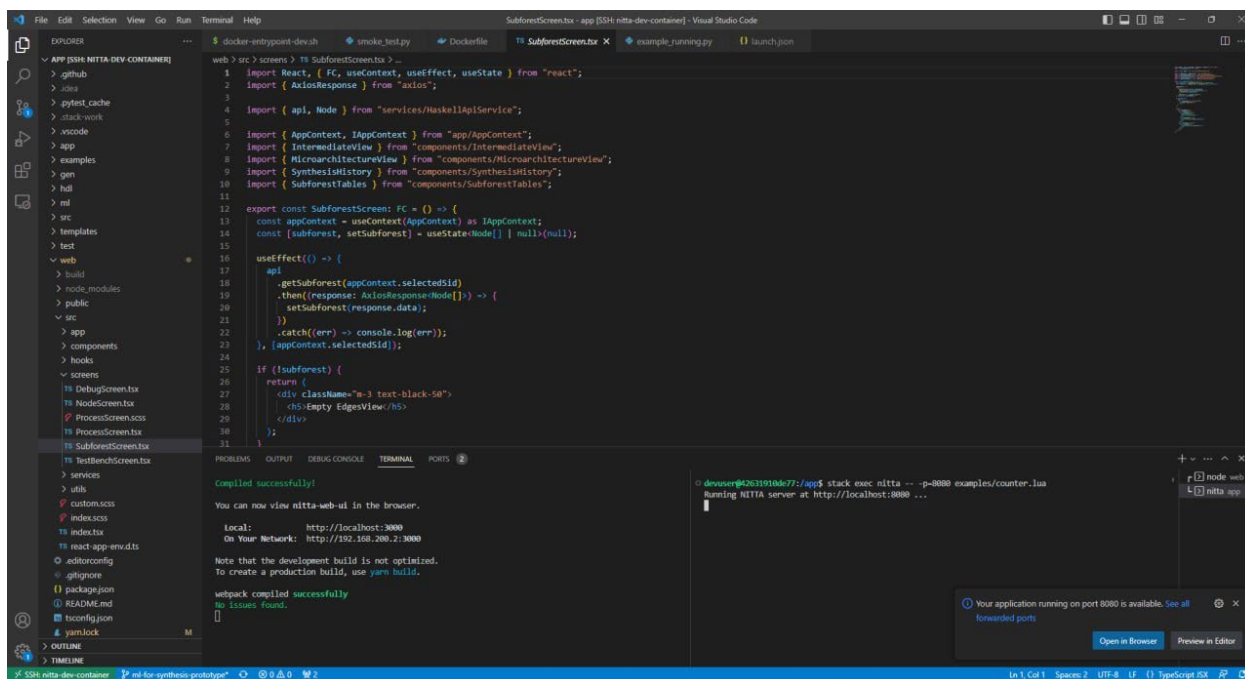


Рисунок 20 – Демонстрация удалённой разработки по SSH в контейнере с использованием Visual Studio Code

Кроме того, протестирован JetBrains Gateway [48], позволяющий получить аналогичный опыт с IDE от компании JetBrains. Исходный код и исполнение – в контейнере, при этом на локальной машине привычный интерфейс. К сожалению, выявлены множество существующих на момент написания дефектов в реализации этого средства. Они предотвращают его эффективное использование, поэтому в дальнейшем разработка на языке Python в рамках настоящей работы также перенесена в VS Code.

Дополнительным преимуществом удалённой IDE как подхода является тот факт, что окружение разработки больше не привязано к физической машине – есть возможность запускать ресурсоёмкие процессы на сервере, работая удалённо с тонкого клиента с доступом к Интернету. Такая практика была неоднократно использована при работе над реализацией усовершенствований в рамках настоящей диссертации.

5.4.2 Написание руководства по эксплуатации для разработчиков

Для других разработчиков САПР НИТТА написано руководство по эксплуатации о том, как пользоваться созданным окружением. Оно описано полностью на английском языке, так всё публичное общение в проекте происходит на нём. Используется язык разметки Markdown для форматирования текста, так как он может отображаться на странице проекта в GitHub.

Добавлены инструкции, учитывающие особенности различных операционных систем, а также общие рекомендации по отладке проблем и описание того, как работают различные компоненты образа. Такая информация должна помочь, если при сборке окружения возникнут ошибки.

5.4.3 Проверка работы контейнера на разных операционных системах

Созданное контейнеризованное рабочее окружение протестировано на различных машинах. Подтверждена работоспособность окружения на различных версиях Windows (10 и 11). Подтверждена работоспособность в виртуальной машине с Linux.

На macOS, к сожалению, выявлены проблемы и со сборкой образа без поддержки GPU. В частности, не было необходимых предкомпилированных пакетов для нужной версии Tensorflow под архитектуру aarch64, используемую на современных устройствах с процессором Apple Silicon. Кроме того, управление правами доступа и пользователями работает на данной ОС по-другому даже в Docker, запускаемом с технической точки зрения в виртуальной машине, что вызвало проблемы с созданием пользователя для работы с Bind mounts. Также возникли проблемы с компиляцией зависимостей NITTA, что привело к необходимости обновить версию GHC в проекте.

5.5 Выводы

Реализовано единое рабочее окружение для САПР специализированных вычислителей NITTA с применением технологий контейнерной виртуализации. Определен необходимый состав окружения, произведено исследование возможностей подхода, реализованы, отлажены и задокументированы сборочные скрипты, предложены готовые средства разработки и совместимые с предлагаемым подходом IDE, решение интегрировано в проект и проверено на различных ОС.

Тестирование образа контейнера на Windows и Linux показало работоспособность всех компонентов (сервера NITTA, веб-интерфейса и модулей машинного обучения), что изображено на рисунке 21.

```
devuser@4bc280c1afab: /app
eliasdev@bunny:~/dev/nitta$ \
> docker run \
  --name=nitta-dev-container \
  --gpus all \
  -p 8888:8888 \
  -p 2222:22 \
  -v"${pwd}:/app" \
  -v"nitta-devuser-home:/home/devuser" \
  -it \
  nitta-dev
SSH into this container if needed for remote debugging in IDEs: ssh -i ml/synthesis/.dev/container_ssh_key -p 2222 -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no devuser@localhost
Jupyter Notebook should be available at: http://localhost:8888/?token=65f22433c68239e99d5e856ac8e67c80b75f432b87153c16
Num of GPUs available to Tensorflow: 1
Available screens (attach with screen -r <name>):
 12.sshd (02/23/23 21:48:14) (Detached)
 16.jupyter-notebook (02/23/23 21:48:14) (Detached)
devuser@4bc280c1afab: /app$ stack exec nitta -- --port=8888 examples/counter.lua
Running NITTA server at http://localhost:8888 ...
```

Рисунок 21 – Запуск контейнера, приветственное сообщение и запуск NITTA

Время автоматической сборки и размер финальных образов составили 30 минут и 8 ГБ для образа без поддержки GPU, а также 45 минут и 17 ГБ для образа с поддержкой GPU. Размер сопоставим с установкой тех же зависимостей вне контейнера, поэтому сочтён допустимым.

К сожалению, на части ОС были выявлены проблемы, что можно объяснить многокомпонентным и сложным, а потому непредсказуемым характером используемых средств. Поставленная задача была в большей степени решена, однако необходимы дальнейшие доработки окружения, посвящённые исправлению проблем на отдельных ОС.

При усовершенствовании прототипа метода синтеза на основе МО в рамках настоящей диссертации удалось успешно применить разработанное контейнеризованное окружение: все действия, связанные с реализацией и испытанием разработанных решений, осуществлялись в созданном Docker-контейнере.

ЗАКЛЮЧЕНИЕ

Успешно спроектирован, реализован и испытан ряд усовершенствований прототипа метода синтеза на основе машинного обучения в САПР специализированных вычислителей НИТТА, а также решён ряд технических вопросов, направленных на инструментальную поддержку и интеграцию в проект разработанных решений. Весь разработанный исходный код опубликован в открытый доступ в репозитории САПР НИТТА [30].

Новый итеративный метод сбора тренировочных данных, согласно результатам испытания [37], позволил повысить генерализуемость моделей МО, использующихся в качестве оценочной функции при синтезе спецвычислителей в САПР НИТТА. Созданной второй версии прототипа метода синтеза на основе МО при испытаниях удалось синтезировать целевые алгоритмы, которые не поддаются синтезу с первой версией.

Новая стратегия обхода дерева синтеза существенно уменьшила среднее время синтеза для многих целевых алгоритмов, в особенности – для сложных алгоритмов с деревьями синтеза большого размера (миллионы вершин и более, ускорение до 6 раз). Негативного влияния на качество результатов синтеза (длительность управляющей программы) обнаружено не было.

В качестве направлений дальнейших исследований можно отметить, прежде всего, повышение репрезентации успешных результатов синтеза в тренировочных данных, собранных с больших деревьев синтеза сложных целевых алгоритмов, а также добавление поддержки в новую стратегию обхода дерева синтеза «далеких» откатов ближе к корню, которые текущая реализация стратегии делать не склонна.

Тем не менее, разработанные усовершенствования значительно повысили эффективность САПР в соответствии с обозначенными критериями. Таким образом, цель работы можно считать достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Liu L. [и др.]. A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications // ACM Computing Surveys (CSUR). 2019. № 6 (52). С. 1–39.
2. Graham P., Nelson B. Reconfigurable processors for high-performance, embedded digital signal processing 1999.С. 1–10.
3. Prakash S. [и др.]. Compact field programmable gate array (FPGA) controller for aircraft/aerospace structures // Proc. of the ICOAST, Bangalore, India. 2008. С. 26–28.
4. Kastensmidt F., Rech P. FPGAs and parallel architectures for aerospace applications // Soft Errors and Fault-Tolerant Design. 2016.
5. Kaarmukilan S. P., Poddar S. [и др.]. FPGA based deep learning models for object detection and recognition comparison of object detection comparison of object detection models using FPGA 2020.С. 471–474.
6. Babu P., Parthasarathy E. Reconfigurable FPGA architectures: A survey and applications // Journal of The Institution of Engineers (India): Series B. 2021. (102). С. 143–156.
7. Branco S., Ferreira A. G., Cabral J. Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey // Electronics. 2019. № 11 (8). С. 1289.
8. Têtu J.-F. [и др.]. A standalone FPGA-Based miner for Lyra2REv2 cryptocurrencies // IEEE Transactions on Circuits and Systems I: Regular Papers. 2020. № 4 (67). С. 1194–1206.
9. Küfeoğlu S., Özkuran M. Bitcoin mining: A global review of energy and power demand // Energy Research & Social Science. 2019. (58). С. 101273.
10. Bai H. [и др.]. Real-time modeling and simulation of electric vehicle battery charger on FPGA 2019.С. 1536–1541.
11. Yang C. [и др.]. Fully integrated FPGA molecular dynamics simulations 2019.С. 1–31.
12. Penskoï A. V. [и др.]. High-level synthesis system based on hybrid reconfigurable microarchitecture // Scientific and Technical Journal of Information Technologies, Mechanics and Optics. 2019. № 2 (19).
13. Бураков И.А. Оптимизация синтеза в САПР специализированных вычислителей с помощью машинного обучения: вып. квал. раб. ... бакалавр: 09.03.04 / Бураков Илья Алексеевич; Национальный исследовательский университет ИТМО. – СПб., 2021. – 54 с.

14. Nane R. [и др.]. A Survey and Evaluation of FPGA High-Level Synthesis Tools // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2016. № 10 (35).
15. Cong J. [и др.]. FPGA HLS today: successes, challenges, and opportunities // ACM Transactions on Reconfigurable Technology and Systems (TRETs). 2022. № 4 (15). С. 1–42.
16. Cong J. [и др.]. High-level synthesis for FPGAs: From prototyping to deployment // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2011. № 4 (30). С. 473–491.
17. Sussmann M., Hill T. Intel HLS Compiler: Fast Design, Coding, and Hardware // White paper. 2017.
18. Siemens EDA Catapult High-Level Synthesis [Электронный ресурс]. URL: https://s3.amazonaws.com/s3.mentor.com/public_documents/datasheet/hls-lp/catapult-high-level-synthesis.pdf (дата обращения: 22.05.2023).
19. Ferrandi F. [и др.]. Bambu: an Open-Source Research Framework for the High-Level Synthesis of Complex Applications IEEE, 2021.С. 1327–1330.
20. Canis A. [и др.]. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems // ACM Transactions on Embedded Computing Systems (TECS). 2013. № 2 (13). С. 1–27.
21. Microchip Technology Inc. Microchip Acquires High-Level Synthesis Tool Provider LegUp to Simplify Development of PolarFire FPGA-based Edge Compute Solutions [Электронный ресурс]. URL: <https://www.microchip.com/en-us/about/news-releases/products/microchip-acquires-high-level-synthesis-tool-provider-legup#> (дата обращения: 17.05.2023).
22. Penskoï A. Synthesis Method for CGRA Processors based on Imitation Model 2021.
23. Бураков, И. А. Оптимизация синтеза в САПР специализированных вычислителей с помощью машинного обучения // Сборник тезисов докладов конгресса молодых ученых. Электронное издание [2021, электронный ресурс]. URL: <https://kmu.itmo.ru/digests/article/7086> (дата обращения: 17.05.2023).
24. Liu H.-Y., Carloni L. P. On learning-based methods for design-space exploration with high-level synthesis 2013.С. 1–7.
25. Carrion Schafer B., Wakabayashi K. Machine learning predictive modelling high-level synthesis design space exploration // IET Computers and Digital Techniques. 2012. № 3 (6).

26. Ferretti L. [и др.]. Graph Neural Networks for High-Level Synthesis Design Space Exploration // ACM Transactions on Design Automation of Electronic Systems. 2023. № 2 (28).
27. Wu N., Xie Y., Hao C. IronMan: GNN-assisted Design Space Exploration in High-Level Synthesis via Reinforcement Learning 2021.
28. Goswami P., Schaefer B. C., Bhatia D. Machine learning based fast and accurate High Level Synthesis design space exploration: From graph to synthesis // Integration. 2023. (88).
29. Taj I., Farooq U. Towards Machine Learning-Based FPGA Backend Flow: Challenges and Opportunities // Electronics (Switzerland). 2023. № 4 (12).
30. Пенской А. [и др.]. Репозиторий САПР NITTA [Электронный ресурс]. URL: <https://github.com/ryukzak/nitta/> (дата обращения: 03.06.2023).
31. Nasteski V. An overview of the supervised machine learning methods // Horizons. b. 2017. (4). С. 51–62.
32. Silver D. [и др.]. Mastering the game of Go with deep neural networks and tree search // Nature. 2016. № 7587 (529).
33. Silver D. [и др.]. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play // Science. 2018. № 6419 (362).
34. Cotarelo A. [и др.]. Improving Monte Carlo Tree Search with Artificial Neural Networks without Heuristics // Applied Sciences. 2021. № 5 (11).
35. Silver D. [и др.]. Mastering chess and shogi by self-play with a general reinforcement learning algorithm // arXiv preprint arXiv:1712.01815. 2017.
36. Kocsis L., Szepesvári C. Bandit based monte-carlo planning 2006.С. 282–293.
37. Бураков И. А. Результаты испытания усовершенствований синтеза [Электронный ресурс]. URL: https://github.com/ryukzak/nitta/blob/4de9180069e48e3657b54fad79f58bfd39417706/evaluation/evaluation_230718_154137.csv (дата обращения: 31.07.2023).
38. Abadi M. [и др.]. TensorFlow: A system for large-scale machine learning 2016.
39. Google LLC TensorFlow Documentation: TensorFlow C API [Электронный ресурс]. URL: https://www.tensorflow.org/install/lang_c (дата обращения: 31.03.2023).
40. Haskell Bindings for TensorFlow Contributors Haskell Bindings for TensorFlow [Электронный ресурс]. URL: <https://github.com/tensorflow/haskell> (дата обращения: 31.03.2023).

41. Python Software Foundation Python 3 Documentation: Embedding Python in Another Application [Электронный ресурс]. URL: <https://docs.python.org/3/extending/embedding.html> (дата обращения: 31.03.2023).
42. Millikin J., Zsigmond A. cpython: Bindings for libcpython [Электронный ресурс]. URL: <https://hackage.haskell.org/package/cpython> (дата обращения: 31.03.2023).
43. Google LLC TensorFlow Serving Documentation [Электронный ресурс]. URL: <https://www.tensorflow.org/tfx/guide/serving> (дата обращения: 31.03.2023).
44. Ramírez S., FastAPI Contributors FastAPI Documentation [Электронный ресурс]. URL: <https://fastapi.tiangolo.com/> (дата обращения: 31.03.2023).
45. HaskellWiki Contributors Bracket pattern // HaskellWiki [Электронный ресурс]. URL: https://wiki.haskell.org/index.php?title=Bracket_pattern&oldid=45883 (дата обращения: 31.03.2023).
46. NVIDIA Corporation. NVIDIA Container Toolkit Documentation [Электронный ресурс]. URL: <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/overview.html> (дата обращения: 24.02.2023).
47. Microsoft Corporation. Visual Studio Code Documentation: Remote Development using SSH [Электронный ресурс]. URL: <https://code.visualstudio.com/docs/remote/ssh> (дата обращения: 18.02.2023).
48. JetBrains s.r.o. JetBrains Gateway [Электронный ресурс]. URL: <https://www.jetbrains.com/remote-development/gateway/> (дата обращения: 18.02.2023).
49. Docker Inc. Docker Documentation: Bind Mounts [Электронный ресурс]. URL: <https://docs.docker.com/storage/bind-mounts/> (дата обращения: 20.02.2023).
50. NVIDIA Corporation. CUDA on WSL User Guide [Электронный ресурс]. URL: <https://docs.nvidia.com/cuda/wsl-user-guide/index.html#getting-started-with-cuda-on-wsl> (дата обращения: 19.02.2023).
51. Microsoft Corporation. Windows AI Documentation: Enable NVIDIA CUDA on WSL [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/windows/ai/directml/gpu-cuda-in-wsl> (дата обращения: 19.02.2023).
52. NVIDIA Corporation. NVIDIA CUDA Toolkit v11.2: Documentation [Электронный ресурс]. URL: <https://docs.nvidia.com/cuda/archive/11.2.0/> (дата обращения: 20.02.2023).
53. Microsoft Corporation. Visual Studio Code Server Documentation [Электронный ресурс]. URL: <https://code.visualstudio.com/docs/remote/vscode-server> (дата обращения: 22.04.2023).